Formal Analysis and Run-time Monitoring of Information Flows in Chromium: Technical Appendix

Lujo Bauer, Shaoying Cai, Limin Jia, Timothy Passaro, Michael Stroucken, and Yuan Tian

February 1, 2015

(*Updated December 1, 2015*)

CMU-CyLab-14-015

CyLab
Carnegie Mellon University
Pittsburgh, PA 15213

Formal Analysis and Run-time Monitoring of Information Flows in Chromium: Technical Appendix[*]

Lujo Bauer, Shaoying Cai[†], Limin Jia, Timothy Passaro, Michael Stroucken, and Yuan Tian
Carnegie Mellon University
[†] Institute for Infocomm Research
{lbauer,liminjia,tpassaro,mxs,yt}@cmu.edu   [†] cais@i2r.a-star.edu.sg

This document is the technical appendix for the following paper:

L. Bauer, S. Cai, L. Jia, T. Passaro, M. Stroucken, and Y. Tian. Run-time monitoring and formal analysis of information flows in Chromium. In *Proceedings of the 22nd Annual Network & Distributed Security Symposium*, February 2015. DOI: 10.14722/ndss.2015.23295

[*]This research was supported in part by US Navy grant N000141310156; NSF grants 0917047, 1018211, and 1320470; and the Singapore National Research Foundation under its International Research Centre @ Singapore Funding Initiative and administered by the IDM Programme Office.

Contents

1 Labels and Policies

We use labels to specify information flow policies. We first introduce the syntax of labels and operations on labels. Then, we discuss how these labels can be used to implement various policies present in browsers today.

1.1 Syntax of Labels

An information flow label, written (S, I, D), is composed of a secrecy label S, an integrity label I, and a declassification label D. The syntactic constructs used in defining labels are summarized below.

Label	ℓ	$::=$	(S, I, D)
Simple Label	κ	$::=$	(σ, ι)
Secrecy label	S	$::=$	$C(\sigma) \mid F(\sigma, \sigma)$
Secrecy tags	σ	$::=$	$\{s_1, \cdots, s_n\}$
Secrecy tag	s	$::=$	$prin \mid url.prin \mid *_u .prin \mid \dagger.prin$
Principal	$prin$	$::=$	$id_{ext} \mid \mathsf{user} \mid *_p$
Integrity label	I	$::=$	$\{i_1, \cdots, i_m\}$
Integrity tag	i	$::=$	API
Declassification tags	D	$::=$	$\{d_1, \cdots, d_n\}$
Declassification tag	d	$::=$	$-s \mid +i \mid s \to s' \mid i \to i'$

Basic labels The basic secrecy label is a set of secrecy tags $\{s_1, \ldots, s_n\}$. Each secrecy tag represents an origin of a secret. We treat the hostname parts of URLs, extension IDs, and the user operating the browser (notated as the tag user) as origins. The integrity label is a set of integrity tags $\{i_1, \ldots, i_n\}$. Each integrity tag represents the privilege to access a sensitive resource (namely, APIs). Even though these tags are reminiscent of permissions, our enforcement mechanism treats them in such a way that it can prevent the privilege escalation that commonly occurs in permission-based systems. The declassification label is a set of capabilities for endorsement ($+i$), declassification ($-s$), and reclassification ($s_1 \to s_2$, $i_1 \to i_2$); we explain these later.

A simple label κ is a pair of a set of secrecy tags σ and a set of integrity tags ι.

Ignoring declassification, an entity labeled $(S_1, I_1, \{\})$ can send data to an entity labeled $(S_2, I_2, \{\})$ if $S_1 \subseteq S_2$ (the destination is authorized for at least as many secrets as the source) and $I_1 \supseteq I_2$ (the source has at least as many "permissions" as the destination).

For example, a DOM subtree that represents content loaded from `ad.com` would have a label that includes the secrecy tag ad[1]; APIs that allow extensions to access the browser's local storage have labels that include a $\mathsf{localStorage}$ integrity tag. Data from a script labeled $(\{\mathsf{cnn}, \mathsf{ad}\}, \{\}, \{\})$ wouldn't be allowed to flow to a DOM node labeled $(\{\mathsf{cnn}\}, \{\}, \{\})$ because the latter is permitted fewer secrets (e.g., isn't permitted secrets labeled ad).

Floating labels The basic secrecy label is too rigid to allow entities to adapt to the browser's dynamic environment. E.g., the label of a DOM node on a `cnn.com` page might initially contain only a cnn secrecy tag to reflect that it contains information from `cnn.com`; after a password manager fills in a form field on the page, however, the label of the form field's DOM node needs to change to reflect that it also contains information from another source.

We express the policy that allows dynamic tainting of an entity as a floating secrecy label. This is similar to JIF's parametric labels. A floating secrecy label, written $F(\sigma_1, \sigma_2)$, has two components: σ_1 is the set of secrecy tags that the entity has at initialization time; σ_2 is a ceiling (upper limit) of the secret that this entity can be tainted with. In other words, a secrecy label $F(\sigma_1, \sigma_2)$ can float to $F(\sigma_1', \sigma_2)$ as long as $\sigma_1' \subseteq \sigma_2$.

This is useful when we want to prevent information from reaching an entity. For example, a floating secrecy label of the form $F(\{\mathsf{siteA}\}, \{\mathsf{siteA}, \mathsf{siteB}\})$ indicates that the labeled entity possesses siteA secrets and is willing to

[1]For brevity, we write ad instead of $\mathsf{ad.com}$ throughout.

receive siteB secrets. Its label will then change to $F(\{\mathsf{siteA}, \mathsf{siteB}\}, \{\mathsf{siteA}, \mathsf{siteB}\})$, continuing to protect the siteB secret. To clearly distinguish floating labels from ordinary ones, we henceforth write non-floating secrecy labels as, e.g., $C(\{\mathsf{siteA}\})$.

Compound labels Going back to the password example, the password field is owned by `cnn.com`, but can be written to by the user or an extension. We would want both labels, as both secrets are involved, but we may want to maintain the notion of a *primary* owner, for purposes that we will shortly show. To this end, we introduce dot-separated compound tags: $s_1.s_2$ indicates that s_1 is the primary owner of the data.

Returning to our example, the secrecy tag cnn.user would be part of the label of a DOM node originally loaded from `cnn.com` (hence cnn) at the user's behest (hence user), e.g., if the tab was opened and the URL typed in by the user. More concretely, nodes in the DOM of the `cnn.com` page would initially be labeled with the secrecy tag $F(\{\mathsf{cnn.user}\}, \{\mathsf{cnn.}*\})$; the $\{\mathsf{cnn.}*\}$ ceiling indicates that it is OK for the node to be tainted (repeatedly) with secrets of all entities whose secrecy label has the form $\{\mathsf{cnn.}*\}$. In contrast, $F(\{\}, \{\dagger.\mathsf{user}\})$ means that an entity with that label can be tainted exactly once, e.g., to $F(\{\mathsf{cnn.user}\}, \{\mathsf{cnn.user}\})$. This label is suitable for content scripts, which are injected into multiple pages, but any script instance is injected into exactly one page.

One purpose for compound labels is to allow labels to reflect which entities influenced the content, while retaining the ability to leave a specific entity in control of the content. For example, we may choose to allow requests to send requests to `cnn.com` only if they are compatible with the destination label $\big(C(\{\mathsf{cnn.}*\}), \{\mathsf{network}\}, \{\}\big)$, which concisely expresses the policy that only `cnn.com` pages are allowed to make requests to `cnn.com`, and that they can do so even if their content has absorbed input from the user or other entities (e.g., they include a secrecy tag like cnn.user). Similar policies can be expressed with declassification, which we discuss next.

Declassification, reclassification, and endorsement Declassification and endorsement capabilities allow an entity (e.g., an extension core) to circumvent constraints that it would otherwise incur because of its secrecy and integrity tags. Declassification is a powerful (and dangerous) operation, and declassification capabilities should be granted to entities only judiciously. At the same time, declassification is necessary, since some extensions, like the password manager, collect many secrets, yet their functionality requires that they (selectively) copy those secrets into arbitrary web pages.

In our password manager example, the $\mathrm{ext}_{\mathrm{pwdMgr}}$ core has the $-*.\mathsf{ext}_{\mathsf{pwd}}$ capability. This is to ensure that no matter how many secrecy tags like $\mathsf{someSite.ext}_{\mathsf{pwd}}$ it accumulates in its secrecy label as a result of saving passwords, it is still able to send data (passwords) to individual web pages (e.g., `cnn.com`). Without declassification, those secrecy tags in $\mathrm{ext}_{\mathrm{pwdMgr}}$'s label would normally cause the label check to fail, since the same tags are not present in `cnn.com`'s secrecy tag, including its ceiling. Declassification (and reclassification and endorsement) are used only when a label check would otherwise fail; they don't affect an entity's secrecy and integrity tags beyond the label check.

Reclassification is a weaker form of declassification: the $s_1 \rightarrow s_2$ reclassification tag indicates that a secrecy tag s_1 can be converted (for the purpose of a label check) to a secrecy tag s_2.

Endorsement tags are similar to declassification tags. To protect the local storage API, we give the API the integrity label $\{\mathsf{localStorage}\}$; only entities that have localStorage in their integrity label, or can add it via endorsement, can use it. Hence, we give the $\mathrm{ext}_{\mathrm{pwdMgr}}$ core the $+\mathsf{localStorage}$ capability, allowing it to elevate its privileges sufficiently to use the local storage API is needed. As with de- and reclassification, endorsement only enables a label check to succeed, and has no persistent effect on the integrity tags in a label.

1.2 Basic Label Operations

Our enforcement mechanism uses a set of label operations to compute labels for components and make policy decisions.

Plain erasure	$C(\sigma)^-$	$=$	σ	$F(\sigma_1,\sigma_2)^-$	$=$	σ_1
	$(S,I,D)^-$	$=$	(S^-,I)			
Floated erasure	$C(\sigma)^*$	$=$	σ	$F(\sigma_1,\sigma_2)^*$	$=$	$\sigma_1 \cup \sigma_2$
	$(S,I,D)^*$	$=$	(S^*,I)			
Taint	$\sigma \rhd_{tnt} C(\sigma')$	$=$	$C(\sigma')$	$\sigma \rhd_{tnt} F(\sigma_1,\sigma_2)$	$=$	$F(\sigma \cup \sigma_1, \sigma_2)$
	$(\sigma,\iota) \rhd_{tnt} (S,I,D)$	$=$	$(\sigma \rhd_{tnt} S, I, D)$			
Merge	$(\sigma_1,\iota_1) \uplus_M (\sigma_2,\iota_2)$	$=$	$(\sigma_1 \cup \sigma_2, \iota_1 \cap \iota_2)$			
Add Ceiling	$(\sigma,\iota)^{FC}$	$=$	$(F(\sigma,\top),\iota,\{\})$			
Generate Lab From URL	$\mathit{labFrom}(prin)$	$=$	$(\{prin\},\{\})$			

The erasure operation ℓ^- removes the declassification capabilities and returns the current secrecy and integrity labels of ℓ. Similar to ℓ^-, function ℓ^* also removes the declassification capabilities from ℓ. The difference is that ℓ^* computes the largest set of secret tags that a component with ℓ can be tainted with.

Next, we define a tainting operation $(\sigma,\iota) \rhd_{tnt} (S,I,D)$ that adds σ to the secrecy tags of S, if S is floating. This operation is only used when the tainting does not exceed the ceiling of S. This operation is used to generate labels for components. For instance, when an event handler receives an event, the event handler's label is tainted with the event's label.

Simple labels form a lattice $(\mathcal{L}, \sqsubseteq)$, where $\mathcal{L}$ is a set of simple labels and $\sqsubseteq$ is a partial order over simple labels. Intuitively, the more secrecy tags a component has, the more secrets it can gather, and the fewer components it can send data to. The fewer integrity tags a component has, the fewer APIs it can access, and the more components it can receive data from. The partial order over simple labels is defined as follows:

Definition 1 (Label order)**.** $(\sigma_1,\iota_1) \sqsubseteq (\sigma_2,\iota_2)$ *iff* $\forall s \in \sigma_1, \exists s' \in \sigma_2$ *s.t.* $s \le s'$ *and* $\iota_2 \subseteq \iota_1$.

For integrity labels, we can do a simple subset comparison. For secrecy tags we use another relation, $s_1 \le s_2$, to compare individual tags. It is defined as follows:

$$s_1 \le s_2 \text{ if } \begin{cases} s_1 = s_2 \\ s_1 = url \text{ and } s_2 \in \{*_u, \dagger\} \\ s_1 = prin \text{ and } s_2 = *_p \\ s_2 = s_a.s_b \text{ and } s_1 \le s_a \\ s_1 = s_a.s_b,\, s_2 = s'_a.s'_b,\, s_a \le s'_a \text{ and } s_b \le s'_b \end{cases}$$

The $\le$ relation is reflexive. The wildcard is higher than a concrete label. s_a is lower than the compound label $s_a.s_b$. We do not include a rule for $s_b \le s_a.s_b$, because a component with label s_b may generate information independent of s_a.

Assume that component A has label ℓ_A and B has ℓ_B. Each time information flows from component A to B, our enforcement mechanism checks whether B's label allows B to learn all the secrets A knows. Formally: ${\ell_A}^- \sqsubseteq {\ell_B}^*$. If the check succeeds, B's label is updated to ${\ell_A}^- \rhd_{tnt} \ell_B$.

Our enforcement mechanism checks labels before web requests or data are sent to remote servers. We define $\mathit{NetDeclassify}(\kappa, url)$ to decide whether data (web requests) with label (σ,ι) can be sent to url as follows.

$$\mathit{NetDeclassify}((\sigma,\iota), url) = \begin{cases} \textsf{allowed} & \text{if } \forall s \in \sigma, s = url.s' \text{ or } url \\ \textsf{disallowed} & \text{otherwise} \end{cases}$$

Declassification capabilities are exercised when components make API calls. We define $\ell \Rightarrow \kappa$ to mean that by raising or declassifying ℓ we can obtain κ.

$$\frac{\ell \Rightarrow_s \kappa \qquad \ell \Rightarrow_i \kappa}{\ell \Rightarrow \kappa}$$

$$\frac{\begin{array}{r}\forall s \in \sigma', \exists d_1, \cdots, d_n \in D \text{ s.t. } \forall j \in [1,n], d_j = s_j \to s_{j+1} \text{ and } s_1 = s, s_{n+1} \in \sigma \\ \text{or } \exists d_1, \cdots, d_n \in D \text{ s.t. } \forall j \in [1,n-1], d_j = s_j \to s_{j+1} \text{ and } s_1 = s, d_n = -s_n\end{array}}{(C(\sigma'), \iota, D) \Rightarrow_s (\sigma, \iota)}$$

$$\frac{\begin{array}{r}\forall s \in \sigma_1, \exists d_1, \cdots, d_n \in D \text{ s.t. } \forall j \in [1,n], d_j = s_j \to s_{j+1} \text{ and } s_1 = s, s_{n+1} \in \sigma \\ \text{or } \exists d_1, \cdots, d_n \in D \text{ s.t. } \forall j \in [1,n-1], d_j = s_j \to s_{j+1} \text{ and } s_1 = s, d_n = -s_n\end{array}}{(F(\sigma_1, \sigma_2), \iota, D) \Rightarrow_s (\sigma, \iota)}$$

$$\frac{\begin{array}{r}\forall i \in \iota, \exists d_1, \cdots, d_n \in D \text{ s.t. } \forall j \in [1,n], d_j = i_j \to i_{j+1} \text{ and } i_{n+1} = i, i_1 \in \iota' \\ \text{or } \exists d_1, \cdots, d_n \in D \text{ s.t. } \forall j \in [2,n], d_j = i_j \to i_{j+1} \text{ and } i_{n+1} = i, d_1 = +i_2\end{array}}{(S, \iota', D) \Rightarrow_i (\sigma, \iota)}$$

1.3 Label Composition

A webpage is composed of components from different origins. Each component comes with its own information flow policy. For instance, an extension's content script's policy is specified in the manifest file of that extension. When the content script is injected into a page, the hosting page has its own information flow policies, which may include policies regarding allowed information flows between the injected script and its surrounding environment. This is where policy composition is necessary.

There are two situations where such policy composition needs to be considered: (1) a host webpage includes external resources such as scripts (including content scripts from extensions) and images; and (2) a host page embeds another page in an iframe or an extension injects an iframe into a host page.

As there are multiple ways to compose policies, we provide a notion of *generalized CSP* (GCSP), which allows a page to specify how to compose the page's policy with that of the external resources. Here, external resources include both page resources and content scripts. We provide four pre-defined composition operations: (1) allow flows allowed by either policy, (2) the page's policy overrides the external resource's policy, (3) the external resource's policy overrides the page's policy, and (4) allow flows allowed by both policies.

GCSP Syntax Each generalized CSP item, denoted χ, maps a principal to a pair of an integrity policy and a natural number indicating which composition rules to use to compute the secrecy label. Below are the definitions.

$$\begin{array}{llcl} \textit{Integrity Pol} & \omega & ::= & \cdot \mid \mathsf{IF}(API_1, \cdots, API_n) \\ & & \mid & \mathsf{IFD}(API_1, \cdots, API_n) \mid \omega_1, \omega_2 \\ \textit{GCSP} & \chi & ::= & \cdot \mid \chi, \textit{prin} \mapsto (\omega, n) \end{array}$$

GCSP uses a natural number to index the compositions. (1) simply takes the union of the secrecy labels, and thus allows the content script to learn the secrets of the DOM and secrets allowed in its manifest file. (2) allows the DOM's policy to override the extension's policy. (3) allows extension's policy to override the DOM's policy. (4) is a strict composition policy that takes the intersection of the policies. If the intersection is empty, we do not inject the script.

The integrity policy is used to specify which interfaces the external scripts can have access too. The effect of $\mathsf{IF}(\mathit{APIs})$ is that the external script has the API names in APIs in its integrity label. This allows the script to access the APIs, but prevents the script from receiving any data from a component that cannot access these APIs. Therefore, we prevent privilege escalation. The effect of $\mathsf{IFD}(\mathit{APIs})$ is that the external script has the endorsement of API names in APIs in its declassification label. This is closer to having the permission to access the APIs. Using endorsement capabilities, the external script can launder data for other components.

Label composition functions Next we formally define the two label composition functions, one for page resources such as content scripts, page scripts, and images; and the other for iframed pages.

$\mathit{computeResLab}(\mathit{url}, \kappa_d, \chi, \ell, \mathit{prin})$ Computes an external resource's label based on url, κ_d, χ, and ℓ. url, κ_d, and χ are the host page/document's URL, simple label, and GCSP respectively. If the external resource is a content script, ℓ denotes the content script's initial label. If the external resource is an image or other object loaded from an external URL, ℓ is the resource label computed from its URL.

The function $\mathit{computeResLab}(\mathit{url}, \kappa_d, \chi, \ell)$ works as follows. Suppose $\kappa_d = (\sigma_d, I_d)$ and $\ell = (F(\sigma_1, \sigma_2), I, D)$. Let (ω, n) denote the composition policy for prin based on χ. The function computes a secrecy label σ' using $\mathit{computeResLab}_{S_n}(\mathit{url}, \sigma_d, F(\sigma_1, \sigma_2))$, an integrity label I' using $\mathit{computeResLab}_{I_n}(\omega, I)$, and an endorsement label D' using $\mathit{computeResLab}_{D_n}(\omega, D)$. The function outputs a label $\ell' = (\sigma', I', D')$.

Before defining $\mathit{computeResLab}_{S_n}(\mathit{url}, \sigma_d, F(\sigma_1, \sigma_2))$, we define an operation that instantiates the secrecy label of a content script to a specific URL. If the external resource is a content script, its initial label will contain some $\dagger.s$ tags. For a secrecy tag with the form $\dagger.s$, the operation $\mathit{url} \succ \dagger.s$ instantiates $\dagger.s$ to $\mathit{url}.s$. For a secrecy tag that does not contain $\dagger$, $\mathit{url} \succ s$ is still s.

$$\mathit{url} \succ s = \begin{cases} \mathit{url}.s' & \text{if } s = \dagger.s' \\ s & \text{otherwise} \end{cases}$$

In the following, we define the function $\mathit{computeResLab}_{S_n}(\mathit{url}, \sigma_d, F(\sigma_1, \sigma_2))$.

Allowed by either	$\mathit{computeResLab}_{S_1}(\mathit{url}, \sigma_d, F(\sigma_1, \sigma_2)) = F(\sigma_1, \sigma_d \cup (\mathit{url} \succ \sigma_2))$
Allowed by CS's manifest	$\mathit{computeResLab}_{S_2}(\mathit{url}, \sigma_d, S) = \mathit{url} \succ S$
Allowed by DOM's label	$\mathit{computeResLab}_{S_3}(\mathit{url}, \sigma_d, F(\sigma_1, \sigma_2)) = F(\cdot, \sigma_d)$
Stricter of CS and DOM	$\mathit{computeResLab}_{S_4}(\mathit{url}, \sigma_d, F(\sigma_1, \sigma_2)) = F(\sigma_1 \cap \sigma_d, \sigma_d \cap (\mathit{url} \succ \sigma_2))$

Then we define the function $\mathit{computeResLab}_{I_n}(\omega, I)$, where $\omega = (\mathsf{IF}(\iota_1), \mathsf{IFD}(\iota_2))$ $= (\mathsf{IF}(API_{a_1}, \cdots, API_{a_n}), \mathsf{IFD}(API_{b_1}, \cdots, API_{b_m}))$.

Allowed by either	$\mathit{computeResLab}_{I_1}(\omega, I) = \mathsf{IF}(\iota_1) \cup I$
Allowed by CS's manifest	$\mathit{computeResLab}_{I_2}(\omega, I) = I$
Allowed by DOM's label	$\mathit{computeResLab}_{I_3}(\omega, I) = \mathsf{IF}(\iota_1)$
Stricter of CS and DOM	$\mathit{computeResLab}_{I_4}(\omega, I) = \mathsf{IF}(\iota_1) \cap I$

Last, we define the function $\mathit{computeResLab}_{D_n}(\omega, D)$, where ω is decomposed as above. We write $+\iota$ to denote the set of endorsement capabilities obtained from the set of integrity tags ι.

Allowed by either	$\mathit{computeResLab}_{D_1}(\omega, D) = +\mathsf{IFD}(\iota_2) \cup D$
Allowed by CS's manifest	$\mathit{computeResLab}_{D_2}(\omega, D) = D$
Allowed by DOM's label	$\mathit{computeResLab}_{D_3}(\omega, D) = +\mathsf{IFD}(\iota_2)$
Stricter of CS and DOM	$\mathit{computeResLab}_{D_4}(\omega, D) = +\mathsf{IFD}(\iota_2) \cap D$

$\mathit{computeFrameLab}(\ell_1, \ell_2, \chi, \mathit{prin})$ Computing a framed page's label based on the frame's policy and the page's policy. The label composition functions for iframed pages are similar. Unlike content scripts, the label does not need to be instantiated by the host page's URL; and therefore the composition function takes the label of the iframe and the label of the embedded page as inputs.

Suppose $\ell_1 = (F(\sigma_1, \sigma_2), I_1, D_1)$, $\ell_2 = (F(\sigma_a, \sigma_b), I_2, D_2)$, then

Allowed by either	$\mathit{computeFrameLab}_{S_1}(F(\sigma_1, \sigma_2), F(\sigma_a, \sigma_b)) = F(\sigma_1 \cup \sigma_a, \sigma_2 \cup \sigma_b)$
Allowed by embedded page's label	$\mathit{computeFrameLab}_{S_2}(F(\sigma_1, \sigma_2), F(\sigma_a, \sigma_b)) = F(\sigma_a, \sigma_b)$
Allowed by parent's policy	$\mathit{computeFrameLab}_{S_3}(F(\sigma_1, \sigma_2), F(\sigma_a, \sigma_b)) = F(\sigma_1, \sigma_2)$
Stricter of the two	$\mathit{computeFrameLab}_{S_4}(F(\sigma_1, \sigma_2), F(\sigma_a, \sigma_b)) = F(\sigma_1 \cap \sigma_a, \sigma_2 \cap \sigma_b)$

Allowed by either	$\mathit{computeFrameLab}_{I_1}(I_1, I_2) = I_1 \cup I_2$
Allowed by embedded page's label	$\mathit{computeFrameLab}_{I_2}(I_1, I_2) = I_2$
Allowed by parent's policy	$\mathit{computeFrameLab}_{I_3}(I_1, I_2) = I_1$
Stricter of the two	$\mathit{computeFrameLab}_{I_4}(I_1, I_2) = I_1 \cap I_2$

The definition of $\mathit{computeFrameLab}_{D_n}$ is the same as $\mathit{computeResLab}_{D_3}(\omega, D)$.

1.4 Policies

Browsers currently implement many security policies. Some of these policies are clearly about information flow and map cleanly to our framework; for others the mapping is less clear. We next revisit several such policies, examining to what extent they map into a framework like ours, as well whether the framework's expressiveness allows richer or more powerful variants of the policies to be stated and enforced.

Same-origin policy Browsers use the same-origin policy (SOP) to manage access to different origins. Origins are usually defined as the tuple (scheme,host,port). Scripts from one origin cannot read content from another origin (e.g., via `XMLHttpRequest`), nor can they locally read data from tabs from other origins. The precise implementation of the SOP is slightly more nuanced: outgoing requests to other origins are allowed, but data that they return to the browser is not forwarded to the entity that initiated the request.

This policy can be easily implemented in our framework. When an entity makes a network request, the label for the network controller is instantiated using the outgoing (scheme,host,port) tuple.[2] For an attempted access to `cnn.com`, this results in the label $\ell_{network} = \big(C(\{\mathsf{cnn.}*\}), \{\mathsf{network}\}, \{\}\big)$. For an entity with label ℓ_e to be allowed to send data on the network, label checks would have to permit a flow from ℓ_e to $\ell_{network}$; this will be allowed only if ℓ_e includes a reclassification or declassification capabilities. To return data from the network, label checks would have to allow the flow from $\ell_{network}$ to ℓ_e. This will succeed only if the secrecy label of ℓ_e contains $\mathsf{cnn.}*$.

In the absence of additional restrictions, the calling page or script could have a sufficiently flexible label ℓ_e to enable either the outgoing or the incoming path. Hence, to enforce the SOP on an entity, the browser needs only to prohibit that entity from having a label that allows it to gather secrecy tags other than those conveying its origin. If we wished to also disallow outgoing cross-origin requests, the browser would need to prevent the entity's label from being able to declassify the tags that describe its origin.

In practice, a strict SOP prevents many commonly used web idioms, which our prototype does not attempt to enforce.

Domain relaxation A page can set its `document.domain` value to a suffix of its current domain, allowing pages with different prefixes of the same hostname to communicate. E.g., a page from `login.a.com` and a page `profile.a.com` can both set their domain to `a.com`, at which point their origins will be considered the same, and the pages will be allowed to access each other's DOM.

Domain relaxation can be implemented in our framework in several ways. One is for `profile.a.com` to have the secrecy tag $F(\{\mathsf{profile.a.com}\}, \{\mathsf{profile.a.com}, \mathsf{login.a.com}\})$, which allows it to receive secrets from `login.a.com`; and for `login.a.com` to have a corresponding secrecy label.

[2] All our hostname-based tags include the scheme and port, though we generally elide this for clarity.

Another option is to give each page the *name*.a.com→a.com reclassification capability. This would allow such pages to talk to a.com, but not yet to each other (because we currently apply reclassification only if necessary to complete a request, and only on the source entity). To accomplish that, their respective secrecy tags *name*.a.com would additionally need to be replaced with a.com, which could be accomplished by the browser crawling over the page's DOM and changing the secrecy tags of any nodes with the appropriate labels from *name*.a.com to a.com.

CSP A CSP allows a page to specify from where page resources (e.g., 3rd-party scripts) can be loaded. The policy applies to images, scripts, etc. CSPs can be broadly interpreted as policies that a host page sets to constraint the information flow between the host page and remote servers from which external resources originate. When the request (e.g., HTTP GET) is sent to a remote server, information flows from the browser to the remote server. The host page can send arbitrary information to the remote server in this way by, e.g., embedding it in the URL string of the HTTP GET request. Once loaded, external resources such as scripts can interact with the rest of the page as well as with remote servers. Our generalized CSP (GCSP) (Section 1.3) can be used to specify the above-mentioned information-flow constraints present in CSPs.

There are two main differences between our GCSP and the existing CSP. First, the existing CSP takes effect only at resource-loading time and does not constrain transitive information flows. E.g., if *url*'s CSP forbids scripts from ad.com, it doesn't mean that an extension's content script running in the same page is prevented from sending to or receiving information from ad.com. GCSP enforces a stricter policy: Any information tagged with a *url* secrecy tag cannot be sent to components that do not have that tag. Second, CSPs also enforce policies other than information flow. For instance, not loading resources from an external resource also prevents the external resource from using local resources such as the screen or CPU. This will effectively protect the user from seeing offensive ads, prevent scripts from draining the laptop battery, etc.

In modern browsers, web pages are allowed to embed third-party content with little restriction. Our modified browser has stricter constraints. To allow web pages to load third-party content, we explicitly enable two-way communication between the page and the external resources.

postMessage postMessage is a JavaScript API which allows web pages to communicate across domains on the client side. postMessage works in two conditions: A parent page embeds another page in an iframe or a parent page opens another page in a new tab. In both cases, the API allows two-way communication. The postMessage send needs to specify the destination, and the receiver can check the source.

To allow communications using postMessage APIs in our system, the sender and receiver's labels need to be adjusted. If a host page were to send data directly to an iframe from a different origin, the request would be denied by our browser. To allow postMessages to work, labels are assigned to the host and iframed page in similar ways as discussed for SOP and CSP.

iframe policies iframes were introduced as an isolation mechanism for a parent page to confine untrusted pages. However, iframes have been abused to embed trusted pages within malicious pages, which then mount phishing and clickjacking attacks.

To prevent such attacks, a server can specify, using the X-Frame-Options header, that the page should not be rendered inside a iframe at all, or should only be rendered inside an iframe of a page from a specified origin.

In a pure information-flow approach, disallowing a page from loading in an iframe cannot easily be done. We can, however, prevent the parent from gaining information from a loaded iframe. For example, if a.com tries to place victim.com in an iframe on its page and receive information from the iframe, it would have to have a secrecy label that can float to include victim.com's secrets. To prevent a.com from having a label that allows this, the browser would have to generate a.com's label from something other than a.com's (self-supplied) CSP. While such restrictions could be expressed cleanly in our framework using composition operators (Section 1.3), we have not yet explored this approach.

Extension host permissions Extensions specify host permissions in its configuration files to ask for permissions to access different pages. The browser matches the URL in the host permissions and the URL of the page to decide whether to inject the script. Our label system can enforce the host permission checking. The label for the content script is formed as $F(\{extension\}, \{extension, host\_permission\})$. When the script is going to be injected, the label check happens and verifies whether the extension is allowed to access the page. We can even do better for blocking information leakage if we use the stricter label composition, in which case if the page is tainted with information from other domains, the content script cannot collect information from other domains.

Extension API permissions Extensions can access some browser APIs if they declare these APIs in the configuration file.

We use integrity labels for controlling access to APIs, which also guards against privilege escalation. If an extension has access to one API, the API will be included in its integrity label, and when information is about to flow from the extension to another party, our system will compare the labels to make sure that the information does not flow to a party which does not have access to that API.

2 System States

Scripts Executable code present in browser extensions as well as webpages is abstractly represented as commands, denoted cmd. We model basic script functionality including updating variables, making function and API calls, and branching on conditions.

$$\begin{array}{llll}
\textit{Command} & cmd & ::= & \mathsf{skip} \mid \mathsf{ret}\ exp \mid x := exp \mid API_a(\vec{exp}) \mid \mathsf{let}\ x = f(exp)\ \mathsf{in}\ cmd \\
 & & \mid & \mathsf{let}\ x = API_s(\vec{exp})\ \mathsf{in}\ cmd \mid cmd_1; cmd_2 \mid \mathsf{if}\ exp\ \mathsf{then}\ cmd_1\ \mathsf{else}\ cmd_2 \\
\textit{Expression} & exp & ::= & x \mid c \mid exp_1\ \mathsf{bop}\ exp_2 \mid \mathsf{uop}\ exp \mid (exp_1, \cdots, exp_n) \\
 & & & \\
\textit{Function decl} & fdecl & ::= & f(x) = cmd \mid x.cmd \\
\textit{Variable Env} & \Gamma & ::= & \cdot \mid \Gamma, x \mapsto v
\end{array}$$

API calls include label checking. Asynchronous API calls are treated as a command $API_a(\vec{exp})$. Synchronous API calls, denoted $\mathsf{let}\ x = API_s(\vec{exp})\ \mathsf{in}\ cmd$, block for return values.

Events and event handlers The syntax of events and event handlers is summarized below. We write e to denote an event, $\mathcal{E}$ to denote an event queue, $EventHandler$ to denote an event handler, and $EventHandlers$ to denote a set of event handlers.

$$\begin{array}{llll}
\textit{Event} & e & ::= & (id_e, eventType, return, info, \kappa) \\
\textit{Event Queue} & \mathcal{E} & ::= & \cdot \mid \mathcal{E} :: e \\
\textit{Return channel} & return & ::= & \mathsf{none} \mid \mathsf{some}(ayncRet, id) \mid \mathsf{some}(e, id) \\
\textit{Event handlers} & EventHandlers & ::= & \cdot \mid EventHandlers, EventHandler \\
\textit{Event handler} & EventHandler & ::= & (id, eventType, x.cmd, \mathcal{E}, BlockingFlag, cmd, id_e, return) \\
\textit{Blocking flag} & BlockingFlag & ::= & \mathsf{blocking} \mid \mathsf{nonblocking}
\end{array}$$

An event is a tuple consisting of a unique event ID (id_e), an event type ($eventType$), whether actions are needed after the event is processed ($return$), additional arguments of the event ($info$), and the information flow label for the event (κ). $eventType$ should contain sufficient information for dispatching an event. For instance, the $eventType$ for a button onClick event is *button1.onclick* and the $eventType$ for a tab onCreated event is tabs.onCreated. $return$ is either none indicating when the event handler for that event finishes, no action needs to be taken; or $\mathsf{some}(ayncRet, id)$ indicating indicating this event is generated in relation to an asynchronous call, and when its event handler finishes processing, the event handler need to invoke a callback function specified by id. $\mathsf{some}(e, id)$ is only used by event handlers and is explained later.

There are two kinds of events in terms of processing modes, namely, blocking events and non-blocking events. For blocking events, event handlers can register to run in blocking mode. The browser finishes processing a blocking event only when all its handlers in blocking mode have been executed. For non-blocking events, handlers cannot run in blocking mode. After dispatching a non-blocking event, the browser can move to the next step without executing all the event handlers. Note that we don't introduce an index to indicate whether an event is blocking or non-blocking. Only the events with certain event types are blocking events, e.g., webRequest.onBeforeRequest events. Given an event, from its type, we can tell whether it is a blocking event. The return channel for an event handler can be $\mathsf{some}(e, id)$ when the event handler is a blocking event handler, processing event e with a unique ID id.

An event handler has its own unique ID, the type of event that it processes, and the code for processing events ($x.cmd$). An event handler can only process one event at a time; events waiting to be processed are stored in an event queue $\mathcal{E}$. The $\mathit{BlockingFlag}$ indicates whether a handler is a blocking event handler. The last three fields in the event handler are the script processing the current event, the ID of the event being processed, and the return information of the event being processed. For page script handlers, id is the node ID of the corresponding script node.

Extensions An extension is a tuple consisting of: a unique ID, one extension core, several content scripts, local storage, an active flag, and a policy label. A static extension core is a tuple consisting of a variable environment Γ, commands cmd corresponding to the main function of the core, and a list of event handlers. A content script contains three identifiers (the ID of the extension it belongs to, its own unique ID, and the ID of the tab in which it runs); programs modeled as Γ, cmd, $\mathit{EventHandlers}$; an index $runat$ for indicating when to inject the script to a tab; and a policy label. The active flag $aFlag$ indicates whether an extension is active.

Installed extension	Ext	$::=$	$(id_{ext}, \mathit{ExtCore}, \mathit{ExtCSs}, \mathit{Storage}, \mathit{activeFlag}, \ell)$
Content scripts	ExtCSs	$::=$	$\cdot \mid \mathit{ExtCSs}, \mathit{ExtCS}$
Content script	ExtCS	$::=$	$(id_{ext}, id_{cs}, id_t, \Gamma, cmd, \mathit{EventHandlers}, runat, \ell)$
Injection time tag	$runat$	$::=$	$\mathsf{DocBegin} \mid \mathsf{DocEnd} \mid \mathsf{DocIdle}$
Extension core	$\mathit{ExtCore}$	$::=$	$(\Gamma, cmd, \mathit{EventHandlers})$
Ext active setting	$\mathit{activeFlag}$	$::=$	$\mathsf{active} \mid \mathsf{inactive}$
Storage	$\mathit{Storage}$	$::=$	$\mathit{objects}, \ell$
Objects	$\mathit{objects}$	$::=$	$\cdot \mid \mathit{objects}, \mathit{object}$
Object	object	$::=$	$(id, \mathit{content})$
Installed extensions	Exts	$::=$	$\cdot \mid \mathit{Ext} :: \mathit{Exts}$

Simplified DOM We model the main page and the iframed subpages contained in a browser tab as a list of documents Docs. A document Doc is defined as $(id_d, url, nodes, \mathit{DocCSs}, \chi, \ell)$. id_d is the document ID. url is the page URL. $nodes$ denotes the page elements. DocCSs are the content scripts injected by extensions. χ denotes the content security policies of the page. Each document is associated with a policy label.

A page consists of many elements, e.g., images, scripts, forms, etc. In the Document Object Model (DOM), the elements in a page are organized in a tree structure. Our model inherits the tree structure from the DOM. The elements in a page are modeled as tree nodes in a document. A node is defined as $(id, attributes, nodes, content, \ell)$. id denotes the node ID. $attributes$ contains general information about the node, e.g., the content type, the URL (if the node loads external object), and the parent node ID, etc. If the node is a script node, $attributes$ also contains the script's event handlers' IDs. $nodes$ are the child nodes. $content$ is a piece of data with a specific format, e.g., an image file. ℓ is the policy label attached on the node.

Documents	$Docs$	$::=$	$\cdot \mid Docs, Doc$
Document	Doc	$::=$	$(id_d, url, nodes, DocCSs, \chi, \ell)$
Node	$node$	$::=$	$(id, attributes, nodes, content, \ell)$
Nodes	$nodes$	$::=$	$\cdot \mid nodes, node$
Attributes	$attributes$	$::=$	$(type, url, \cdots)$
Content Type	$type$	$::=$	stylesheet $\mid$ script $\mid$ image $\mid$ $\cdots$

Bookmarks Like the DOM, bookmarks have a tree structure. Operations on bookmarks include insertion, deletion, and mutation of nodes and subtrees. As with the DOM, we could allow each node to be tainted with the label of the entity that updates the data structure. The drawback is that to prevent information leakage, many simple operations would be prohibited. For instance, if a script with many secrets wrote to the root of the bookmark tree, then no entities that are allowed fewer secrets could read any bookmark. Since bookmarks have a long life cycle, this is too prohibitive. Instead, we borrow ideas from multi-level secure execution. We implement a multi-level bookmark $MBookmarks$ data structure, consisting of a set of pairs of a bookmark $bookmark$ and a simple label κ. The label indicates the secrecy and integrity level of the bookmark. A bookmark is a tree: each leaf node is a bookmark entry and each non-leaf node represents a directory.

Multi-level bookmarks	$MBookmarks$	$::=$	$\cdot \mid MBookmarks, MBookmark$
Bookmarks	$MBookmark$	$::=$	$bookmarks, \kappa$
Bookmarks	$bookmarks$	$::=$	$\cdot \mid bookmarks, bookmark$
Bookmark	$bookmark$	$::=$	$(id, title, bookmarks \mid id, title, url)$

Cookies Cookies are similar to bookmarks in that they are long lived; tainting them would interfere with normal functionality. Hence, we label each with a simple label κ. For cookies, the label corresponds to the cookie's domain, so web sites can set and retrieve their cookies, which is the main functionality needed of cookies. To operate on cookies, an entity needs to be able to reclassify to the secrecy label of a cookie's domain, which is consistent with having the ability to access content from that domain.

Cookies	$Cookies$	$::=$	$\cdot \mid Cookies, Cookie$
Cookie	$Cookie$	$::=$	$(name, value, url, \kappa)$

Histories History entries have simple labels as well. Each history item $history$ has the secrecy and integrity label of the entity that caused the history entry to be created. When querying history, an entity with label ℓ is given results composed of entries whose label is lower than or equal to ℓ. When deleting history entries, only entries with label equal to or higher than ℓ are removed.

Histories	$histories$	$::=$	$\cdot \mid histories, history$
History	$history$	$::=$	$(id, url, name, visitTime, visitType, \kappa)$

Runtime extension instances When an extension core injects a content script using API chrome.tabs.executeScript, the injected script may not run right away. Instead, it is stored in $progInjCSs$. $DocCSs$ is the list of active content scripts. id_r is the unique identifier for that runtime instance. The runtime instance of an extension core is denoted $ExtCoreR$.

Injected content scripts	$progInjCSs$	$::=$	$\cdot \mid progInjCSs, ExtCS$
Doc content scripts	$DocCSs$	$::=$	$\cdot \mid DocCSs, DocCS$
Doc content script	$DocCS$	$::=$	$(id_{ext}, id_{cs}, id_r, \Gamma, cmd, EventHandlers, \ell)$
Runtime Core	$ExtCoreR$	$::=$	$id_{ext}, ExtCore, \ell$
Extension cores	$ExtCoreRs$	$::=$	$\cdot \mid ExtCoreRs, ExtCoreR$

Browser state The top-level system state Σ contains tabs in the browser ($Tabs$), run-time extension cores ($ExtCoreRs$), static copies of programmatically injected content scripts ($progInjCSs$) installed extensions ($Exts$), cookies ($Cookies$), bookmarks ($MBookmarks$), histories ($histories$), and user actions (UI).

System state	Σ	::=	$(\Psi, Tabs, ExtCoreRs, progInjCSs, Exts,$ $Cookies, MBookmarks, histories, UI)$
Browser state	Ψ	::=	$\cdot \mid \Psi, \psi$
Async Call Browser States	$ayncCall$	::=	chrome.management.setEnabled $\mid \cdots$
Async Call Ret States	$ayncRet$	::=	chrome.runtime.sendMessage.ResponseGenerated $\mid \cdots$
Browser sub-state	ψ	::=	ws.beforeRequest$(\kappa, id_t, id_d, id_n, id_e, url, info) \cdots$
		$\mid$	$ayncRet \mid ayncCall$
		$\mid$	DoneBlkEvtState$(\cdots) \mid$ ProcBlkEvtState$(\cdots)$
Tabs	$Tabs$	::=	$\cdot \mid Tabs, Tab$
Tab	Tab	::=	$(id_t, Docs, url, EventHandlers, \ell)$
UI	UI	::=	(user, $cmd, \ell)$

User is denoted by a tuple consisting of a unique ID user, commands that user intend to executes, and the label of the user. For instance, the API call chrome.tabs.create$(\cdots)$ corresponds to a user pressing "Ctrl + T".

A browser tab (tab) is the tuple $(id_t, Docs, url, EventHandlers, \ell)$. Each tab has a unique ID, id_t. $Docs$ denotes the documents in the tab, including the top-level document and sub-documents (if any). $EventHandlers$ comes from the page scripts. When a script node is loaded, we extract the event handlers and add them to $ScriptHandlers$. Later, even the script node is removed from the doc, the handlers are still kept in $ScriptHandlers$. $\mathcal{E}$ are the DOM events generated in the tab. ℓ is the tab's label.

The initial system state is denoted Σ_{init}. When a system starts from a clean state (not recovered from a pre-stored state), it does not contain any tab, runtime extension cores, or programmatically injected content scripts. So Σ_{init} is "$\cdot, \cdot, \cdot, Exts, MCookies, MBookmarks, UI$".

Ψ denotes the browser's state. This can be composed of several types of sub-state.

The first type is the web request internal state. For instance, if the browser is going to load an external image to a page, the system will generate a ws.beforeRequest state. A ws.beforeRequest takes seven arguments: κ, id_t, id_d, id_n, id_e, url, and $info$. κ is the web request issuer's label. In the above example, the issuer is the DOM node which is going to load the image. id_t, id_d, and id_n are the hosting tab, doc and node ID respectively. url is the image's URL. $info$ is a reserved place for storing additional information. For example, if the web request is for loading a subpage into an iframe, we could store the frame policy for the subpage in $info$.

The second type is the asynchronous API call state. When an entity makes an asynchronous API call, a corresponding asynchronous call state $ayncCall$ will be generated. After the browser processes the API call, if there is a return, an asynchronous call return state $ayncRet$ will be generated. $ayncRet$ contains the API's return value.

The third type is the blocking event state. Given a blocking event, if there are multiple matching blocking event handlers, the browser can only proceed to the next step if all the handlers have finished processing the event. Our model keeps track of event processing. When a blocking event is dequeued from a blocking event handler, a ProcBlkEvtState$(\cdots)$ is generated. When the blocking handler finishes processing the event, a DoneBlkEvtState$(\cdots)$ state is generated. With these states, we know whether a blocking event has been processed by all the matching blocking handlers.

Evaluation contexts We first define script variable context environments as follows.

Docs script env	$Docs_{\mathit{env}}(id)$	::=	$\cdot \mid Docs :: Doc_{\mathit{env}}(id)$
Doc script env	$Doc_{\mathit{env}}(id)$	::=	$id_d, url, nodes_{\mathit{env}}(id), ExtCSs, \ell$
Node script env	$node_{\mathit{env}}(id)$	::=	$id, attributes, nodes, content, [\![\ ]\!], \ell$
Nodes script env	$nodes_{\mathit{env}}(id)$	::=	$\cdot \mid nodes :: node_{\mathit{env}}(id)$

These contexts include a hole $\llbracket\;\rrbracket$ in the place of the variable environments (Γ). The purpose of the variable context is to identify such Γs so values of variables can be looked up and the environment itself can be updated.

We define the execution contexts, which contain a hole $\llbracket\;\rrbracket$indicating the position of the current evaluation.

State context	$\Sigma_{ctx}(id)$	$::=$	$\Psi, Tabs_{ctx}(id), ExtCoreRs, progInjCSs, Exts, Cookies,$ $MBookmarks, histories, UI, sMode$
		$\mid$	$\Psi, Tabs, ExtCoreRs_{ctx}(id), progInjCSs, Exts, Cookies,$ $MBookmarks, histories, UI, sMode$
Tabs context	$Tabs_{ctx}(id)$	$::=$	$\cdot \mid Tabs :: Tab_{ctx}(id)$
Tab context	$Tab_{ctx}(id)$	$::=$	$id_t, Docs_{ctx}(id), url, EventHandlers, \ell$
		$\mid$	$id_t, Docs_{env}(id), url, EventHandlers_{ctx}(id), \ell$
Doc content scripts context	$DocCSs_{ctx}(id)$	$::=$	$\cdot \mid DocCSs :: DocCS_{ctx}(id)$
Doc content script context	$DocCS_{ctx}(id_{cs})$	$::=$	$id_{ext}, id_{cs}, id_t, \llbracket\;\rrbracket, cmd_{ctx}, EventHandlers, \ell$
		$\mid$	$id_{ext}, id_{cs}, id_t, \llbracket\;\rrbracket, cmd, EventHandlers_{ctx}(id), \ell$
Docs context	$Docs_{ctx}(id)$	$::=$	$\cdot \mid Docs :: Doc_{ctx}(id)$
Doc context	$Doc_{ctx}(id)$	$::=$	$id_d, url, nodes, DocCSs_{ctx}(id), \ell$
Event handlers context	$EventHandlers_{ctx}(id)$	$::=$	$\cdot \mid EventHandlers :: EventHandler_{ctx}(id)$
Event handler context	$EventHandler_{ctx}(id)$	$::=$	$id, eventType, x.cmd, \mathcal{E}, cmd_{ctx}, id_e, BlockingFlag, return$
Extension cores context	$ExtCoreRs_{ctx}(id)$	$::=$	$\cdot \mid ExtCoreRs :: ExtCoreR_{ctx}(id)$
Extension core context	$ExtCoreR_{ctx}(id_{ext})$	$::=$	$id_{ext}, (\llbracket\;\rrbracket, cmd_{ctx}, EventHandlers), \ell$
		$\mid$	$id_{ext}, (\llbracket\;\rrbracket, cmd, EventHandlers_{ctx}), \ell$
Command context	cmd_{ctx}	$::=$	$\llbracket\;\rrbracket \mid \mathsf{let}\ x = \llbracket\;\rrbracket\ \mathsf{in}\ cmd \mid cmd_{ctx}; cmd$
UI context	$UI_{ctx}(\mathsf{user})$	$::=$	$\mathsf{user}, x.cmd, cmd_{ctx}, \ell$

We write $\Sigma_{ctx}\llbracket x \rrbracket_{(id)}$ to denote the state resulting from plugging construct x into context $\Sigma_{ctx}(id)$. In terms of script context, we plug in two constructs, $\Sigma_{ctx}\llbracket \Gamma, x \rrbracket_{(id)}$, where Γ is the context that maps global variables to their values. We define a function $\mathsf{ctxOfId}(\Sigma, id)$ to return the label of the closest enclosing context of the element with identifier id in the system state Σ.

3 Transition Rules

The top-level transition rules are of the form $\Xi; \Sigma; \mathcal{E} \xrightarrow{\beta} \Xi'; \Sigma'; \mathcal{E}'$. Here, Ξ denotes remote servers, which are active entities that exchange information with the browser. Σ is the browser state. $\mathcal{E}$ denotes events waiting to be processed. Events can be user inputs, API requests, and other internal browser events. Each transition is labeled with an action β, representing the observable effects of that transition. In this technical report, for rules where Ξ stay the same, we omit them from the rules.

3.1 Auxiliary Definitions

Web servers The purpose of modeling the web servers is to model web attackers. Each web server is a pair of its URL and program. A web server either listens to a request (listen), or sends a header to the browser ($\mathsf{sendHeader}(h)$), or sends content to the browser ($\mathsf{sendContent}(h)$).

Web servers	Ξ	$::=$	$\cdot \mid \Xi, (url, \mathrm{exp})$
Server Program	exp	$::=$	$\mathsf{listen}; x.\mathrm{exp} \mid \mathsf{sendHeader}(h); \mathrm{exp} \mid \mathsf{sendContent}(h); \mathrm{exp} \mid \mathsf{skip}$

Actions Observable actions, denoted α, include API calls, invocations of callbacks, and processed events. These actions pass on information to event handlers. The browser makes internal transitions, which do not produce observable effects. We use τ to label such transitions, and call them silent transitions. We define an execution trace ρ as the sequence of non-silent actions in a transition sequence.

$$\begin{array}{llll} \textit{Actions} & \alpha & ::= & e \mid \textit{API}(\kappa_s, \textit{args}) \mid \mathsf{Ret}(\kappa_s, \textit{args}) \\ \textit{Generalized Action} & \beta & ::= & \alpha \mid \tau \\ \textit{Traces} & \rho & ::= & \epsilon \mid \rho, \alpha \end{array}$$

Definition of enqueuing an event We define $e \sim_t \textit{Tab}$ to mean that event e is related to the tab $\textit{Tab}$. We assume that each event carries the identifier of the tab that it is generated from. For events that are returns to asynchronous API calls, it is related to a tab if that tab contains an event handler which is the callback function of that event. This can be checked by examining the event type. An event will be added to each matching event handler's event queue. Labels are checked when an event handler is about to process the event. Events are not propagated across frames.

$$\textit{EventHandler} \lhd_Q e = \\ \begin{cases} \textit{EventHandler} & \textit{eventTypeOf}(\textit{EventHandler}) \neq \textit{eventTypeOf}(e) \\ \textit{EventHandler}[\textit{eventQueue} \Leftarrow \textit{eventQueue} :: e] & \textit{eventTypeOf}(\textit{EventHandler}) = \textit{eventTypeOf}(e) \end{cases}$$

We define $\textit{EventHandlers} \lhd_Q e$ to be the lifting of $\textit{EventHandler} \lhd_Q e$ to the list of event handlers.

$$\begin{array}{lcl} \textit{ExtCoreR} \lhd_Q e & = & \textit{ExtCoreR}[\textit{EventHandlers} \Leftarrow \textit{EventHandlers} \lhd_Q e] \\ \textit{DocCS} \lhd_Q e & = & \textit{DocCS}[\textit{EventHandlers} \Leftarrow \textit{EventHandlers} \lhd_Q e] \\ & & \\ \textit{Doc} \lhd_{cs} e & = & \textit{Doc}[\textit{DocCSs} \Leftarrow \textit{DocCSs} \lhd_Q e] \\ \textit{Tab} \lhd_{ps} e & = & \begin{cases} \textit{Tab}[\textit{EventHandlers} \Leftarrow \textit{EventHandlers} \lhd_Q e] & e \sim_t \textit{Tab} \\ \textit{Tab} & \text{otherwise} \end{cases} \\ & & \\ \textit{Tab} \lhd_{cs} e & = & \begin{cases} \textit{Tab}[\textit{Docs} \Leftarrow \textit{Docs} \lhd_{cs} e] & e \sim_t \textit{Tab} \\ \textit{Tab} & \text{otherwise} \end{cases} \\ & & \\ \textit{Tab} \lhd_Q e & = & \textit{Tab} \lhd_{cs} e \lhd_{ps} e \end{array}$$

Relations between internal browser state and events For web requests, the browser processes a sequence of events in a particular order. We use internal browser state to track such ordering; an event in this sequence can only be processed when the browser is in a corresponding state.

We define two relations to specify the correspondence between an internal browser state and an event: $\psi \sim_b e$ and $\psi \sim_{nb} e$. Because events can be blocking or nonblocking, $\sim_b$ relates a state and a blocking event and $\sim_{nb}$ relates a state and a non-blocking event. We list the elements in both relations below.

$\mathsf{ws.beforeRequest}(\kappa, id_{cb}, (id_t, id_d, id_n), id_e, url, info) \sim_b (id_e, \mathsf{webRequest.onBeforeRequest}, \cdots, \kappa)$
$\mathsf{ws.beforeSendHeader}(\kappa, id_{cb}, (id_t, id_d, id_n), id_e, url, info) \sim_b (id_e, \mathsf{webRequest.onBeforeSendHeaders}, \cdots, \kappa)$
$\mathsf{ws.headerReceived}(\kappa, id_{cb}, (id_t, id_d, id_n), id_e, url, info) \sim_b (id_e, \mathsf{webRequest.onHeadersReceived}, \cdots, \kappa)$

$\mathsf{ws.beforeRequest.redirect}(\kappa, id_{cb}, (id_t, id_d, id_n), id_e, url, info) \sim_{nb} (id_e, \mathsf{webRequest.onBeforeRedirect}, \cdots, \kappa)$
$\mathsf{ws.headerSent}(\kappa, id_{cb}, (id_t, id_d, id_n), id_e, url, info) \sim_{nb} (id_e, \mathsf{webRequest.onSendHeaders}, \cdots, \kappa)$
$\mathsf{ws.responseStarted}(\kappa, id_{cb}, (id_t, id_d, id_n), id_e, url, info) \sim_{nb} (id_e, \mathsf{webRequest.onResponceStarted}, \cdots, \kappa)$
$\mathsf{ws.completed}(\kappa, id_{cb}, (id_t, id_d, id_n), id_e, url, info, (content, \kappa_1)) \sim_{nb} (id_e, \mathsf{webRequest.onCompleted}, \cdots, \kappa)$
$\mathsf{ws.failed}(\kappa, id_{cb}, (id_t, id_d, id_n), id_e, url, info, \ell) \sim_{nb} (id_e, \mathsf{webRequest.onErrorOccurred}, \cdots, \kappa)$
$\mathsf{readytoCreateDoc}(id_t, id_d, id_e, content, \ell_{doc}) \sim_{nb} (id_e, \mathsf{webNavigation.onCommitted}, \cdots, \kappa)$

Each web request state contains several IDs. If a web request state is for loading external object to a node or doc, id_{cb} is set as $\cdot$ and (id_t, id_d, id_n) stores tab ID, Doc ID and node ID. If a web request state is from a direct web request API call, id_{cb} is the callback handler's event type, (id_t, id_d, id_n) is set to null.

3.2 Browser Internal State Transition Rules

A browser can change its internal state. We define two state transition functions to specify to which state the browser should transition based on the current internal state. The first function $\mathsf{nextState}(\psi, arg)$ takes the current state and an auxiliary argument which could be void or $\mathsf{noChange}$ or $\mathsf{Blocked}(c)$ as inputs, and returns a new state and a list of events. Generally, the second argument is set as void. However, when processing a blocking event, the browser's next state depends on the event handlers' return value as well. If there is no blocking event handler for the event, the second argument is set as $\mathsf{noChange}$; else, it is set as $\mathsf{Blocked}(c)$ where c is the return value. This function is used in the sequence of transitions that a browser takes to process web requests.

The second function $\mathsf{nextStateC}(\Sigma, \psi)$ takes the entire browser state and the current browser state as arguments, and returns an updated browser state and a list of events. This function is used when the internal transition also alters other pieces of the browser state such as the DOM.

$$\frac{\begin{array}{l}\Sigma = (\Psi, \cdots) \qquad \Psi = \Psi' :: \psi \\ (\psi', \mathcal{E}_1) = \mathsf{nextState}(\psi, \mathsf{void}/\mathsf{noChange}/\mathsf{Blocked}(c)) \qquad \Sigma' = \Sigma[\Psi \Leftarrow \Psi' :: \psi']\end{array}}{\Sigma; \mathcal{E} \xrightarrow{\tau} \Sigma'; \mathcal{E} :: \mathcal{E}_1} \text{ INTERNALSTATETRANSITION1}$$

$$\frac{\Sigma = (\Psi, \cdots) \qquad \Psi = \Psi' :: \psi \qquad (\Sigma', \mathcal{E}_1) = \mathsf{nextStateC}(\Sigma, \psi)}{\Sigma; \mathcal{E} \xrightarrow{\tau} \Sigma'; \mathcal{E} :: \mathcal{E}_1} \text{ INTERNALSTATETRANSITION2}$$

3.3 Script Transition Rules

Beta rules for scripts Given f, id, and ℓ, $\mathsf{getFunction}$ will find the code of the function f in the matching scope, and execute it. The content script run at the end of Doc loading can refer to the global variables and functions defined in the content scripts run at the beginning of Doc loading, but not vise versa.

$$\frac{\Gamma, e \Downarrow v}{\Gamma, x := e \rightarrow_\beta \Gamma[x \mapsto v], \mathsf{skip}} \text{ ASSIGN} \qquad \frac{\Gamma, e \Downarrow \mathsf{true}}{\Gamma, \mathsf{if}\ e\ \mathsf{then}\ cmd_1\ \mathsf{else}\ cmd_2 \rightarrow_\beta \Gamma, cmd_1} \text{ IF-T}$$

$$\frac{\Gamma, e \Downarrow \mathsf{false}}{\Gamma, \mathsf{if}\ e\ \mathsf{then}\ cmd_1\ \mathsf{else}\ cmd_2 \rightarrow_\beta \Gamma, cmd_2} \text{ IF-F} \qquad \frac{}{\Gamma, \mathsf{skip}; cmd \rightarrow_\beta \Gamma, cmd} \text{ SKIP}$$

$$\frac{}{\Gamma, \mathsf{let}\ x = v\ \mathsf{in}\ cmd \rightarrow_\beta \Gamma, cmd[v/x]} \text{ LET} \qquad \frac{x.cmd = \mathsf{getFunction}(\Gamma, f, id)}{\Gamma, f(\vec{e}) \rightarrow_\beta \Gamma, cmd[\vec{e}/x]} \text{ FUNCTIONCALL}$$

Top-level beta

$$\frac{\Gamma, cmd \rightarrow_\beta \Gamma', cmd'}{\Sigma_{ctx}[\![\, \Gamma, cmd \,]\!]_{id}; \mathcal{E} \xrightarrow{\tau} \Sigma_{ctx}[\![\, \Gamma', cmd' \,]\!]_{id}; \mathcal{E}} \text{ CONTEXT}$$

Returns The last instruction of an event handler is the return instruction. If the return field of the event handler is none, indicating that no one is waiting for the return value of this handler, then the script evaluates to skip. The three rules RETURN-N-CORE, RETURN-N-CS, and RETURN-N-PAGE are for core scripts, content scripts, and page scripts respectively.

$$\frac{\begin{array}{l}\Sigma_{ctx}(id) = (\Psi, Tabs, ExtCoreRs_{\mathbf{ctx}}(id), \cdots) \qquad ExtCoreR_{\mathbf{ctx}}(id) \in ExtCoreRs_{\mathbf{ctx}}(id) \\ ExtCoreR_{\mathbf{ctx}}(id) = (id, \_, EventHandlers :: EventHandler_{\mathbf{ctx}}, \ell) \\ EventHandler_{\mathbf{ctx}}(id) = (\cdots, \mathsf{nonblocking}, \mathsf{none})\end{array}}{\Sigma_{ctx} \llbracket \Gamma, \mathsf{ret}\ v \rrbracket_{id}; \mathcal{E} \xrightarrow{\tau} \Sigma_{ctx} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id}; \mathcal{E}} \text{ RETURN-N-CORE}$$

$$\frac{\begin{array}{l}\Sigma_{ctx}(id) = (\Psi, Tabs_{\mathbf{ctx}}(id), \cdots) \\ t_{\mathbf{ctx}}(id) \in Tabs_{\mathbf{ctx}}(id) \qquad t_{\mathbf{ctx}}(id) = (\cdots, Docs_{\mathbf{ctx}}(id), \cdots) \qquad Doc_{\mathbf{ctx}}(id) \in Docs_{\mathbf{ctx}}(id) \\ Doc_{\mathbf{ctx}}(id) = (\cdots, DocCSs_{\mathbf{ctx}}(id), \cdots) \qquad DocCS_{\mathbf{ctx}}(id) \in DocCSs_{\mathbf{ctx}}(id) \\ DocCS_{\mathbf{ctx}}(id) = (\_, id, \cdots, EventHandlers :: EventHandler_{\mathbf{ctx}}(id), \ell,) \\ EventHandler_{\mathbf{ctx}}(id) = (\cdots, \mathsf{none})\end{array}}{\Sigma_{ctx} \llbracket \Gamma, \mathsf{ret}\ v \rrbracket_{id}; \mathcal{E} \xrightarrow{\tau} \Sigma_{ctx} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id}; \mathcal{E}} \text{ RETURN-N-CS}$$

$$\frac{\begin{array}{l}\Sigma_{ctx}(id) = (\Psi, Tabs_{\mathbf{ctx}}(id), \cdots) \qquad t_{\mathbf{ctx}}(id) \in Tabs_{\mathbf{ctx}}(id) \\ t_{\mathbf{ctx}}(id) = (id_t, Docs(id), url, EventHandlers :: EventHandler_{\mathbf{ctx}}(id) \cdots) \\ Doc(id) \in Docs(id) \qquad Doc(id) = (\cdots, nodes(id), \cdots) \qquad node(id) \in nodes(id) \\ node(id) = (id, attributes, nodes_1, content, \llbracket\, \rrbracket, \ell) \qquad EventHandler_{\mathbf{ctx}}(id) = (\cdots, \mathsf{none})\end{array}}{\Sigma_{ctx} \llbracket \Gamma, \mathsf{ret}\ v \rrbracket_{id}; \mathcal{E} \xrightarrow{\tau} \Sigma_{ctx} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id}; \mathcal{E}} \text{ RETURN-N-PAGE}$$

The next three rules RETURN-CORE, RETURN-CS, and RETURN-PAGE are for core scripts, content scripts, and page scripts to return to callback functions respectively. The $return$ of the event handler is $\mathsf{some}(R, id_r)$, indicating that there is a callback function waiting for the completion of this event handler. Here, the event handler is actually an asynchronous API call (or a callback), R is the name of the original asynchronous API call. When the event handler returns, the internal browser state is updated to include a new state $ayncAPIRetState(R, \ell^-, id_r, v)$, stating that the asynchronous API call to R has completed with return value v, label ℓ^-, and the callback's ID is id_r. We will show an example at the end of this section.

$$\frac{\begin{array}{l}\Sigma_{ctx}(id) = (\Psi, \mathit{Tabs}, \mathit{ExtCoreRs}_{\mathbf{ctx}}(id), \cdots) \qquad \mathit{ExtCoreR}_{\mathbf{ctx}}(id) \in \mathit{ExtCoreRs}_{\mathbf{ctx}}(id) \\ \mathit{ExtCoreR}_{\mathbf{ctx}}(id) = (id, \_, \mathit{EventHandlers} :: \mathit{EventHandler}_{\mathbf{ctx}}, \ell) \\ \mathit{EventHandler}_{\mathbf{ctx}}(id) = (\cdots, \mathsf{nonblocking}, \mathsf{some}(R, id_r)) \\ \psi = \mathit{ayncAPIRetState}(R, \ell^-, id_r, v) \qquad \Sigma'_{ctx}(id) = \Sigma_{ctx}(id)[\Psi \Leftarrow \Psi :: \psi]\end{array}}{\Sigma_{ctx} \| \Gamma, \mathsf{ret}\ v \|_{id}; \mathcal{E} \xrightarrow{\tau} \Sigma'_{ctx} \| \Gamma, \mathsf{skip} \|_{id}; \mathcal{E}} \text{ RETURN-CORE}$$

$$\frac{\begin{array}{l}\Sigma_{ctx}(id) = (\Psi, \mathit{Tabs}_{\mathbf{ctx}}(id), \cdots) \\ t_{\mathbf{ctx}}(id) \in \mathit{Tabs}_{\mathbf{ctx}}(id) \qquad t_{\mathbf{ctx}}(id) = (\cdots, \mathit{Docs}_{\mathbf{ctx}}(id), \cdots) \qquad \mathit{Doc}_{\mathbf{ctx}}(id) \in \mathit{Docs}_{\mathbf{ctx}}(id) \\ \mathit{Doc}_{\mathbf{ctx}}(id) = (\cdots, \mathit{DocCSs}_{\mathbf{ctx}}(id), \cdots) \qquad \mathit{DocCS}_{\mathbf{ctx}}(id) \in \mathit{DocCSs}_{\mathbf{ctx}}(id) \\ \mathit{DocCS}_{\mathbf{ctx}}(id) = (\_, id, \cdots, \mathit{EventHandlers} :: \mathit{EventHandler}_{\mathbf{ctx}}(id), \ell,) \\ \mathit{EventHandler}_{\mathbf{ctx}}(id) = (\cdots, \mathsf{some}(R, id_r)) \\ \psi = \mathit{ayncAPIRetState}(R, \ell^-, id_r, v) \qquad \Sigma'_{ctx}(id) = \Sigma_{ctx}(id)[\Psi \Leftarrow \Psi :: \psi]\end{array}}{\Sigma_{ctx} \| \Gamma, \mathsf{ret}\ v \|_{id}; \mathcal{E} \xrightarrow{\tau} \Sigma'_{ctx} \| \Gamma, \mathsf{skip} \|_{id}; \mathcal{E}} \text{ RETURN-CS}$$

$$\frac{\begin{array}{l}\Sigma_{ctx}(id) = (\Psi, \mathit{Tabs}_{\mathbf{ctx}}(id), \cdots) \qquad t_{\mathbf{ctx}}(id) \in \mathit{Tabs}_{\mathbf{ctx}}(id) \\ t_{\mathbf{ctx}}(id) = (id_t, \mathit{Docs}(id), \mathit{url}, \mathit{EventHandlers} :: \mathit{EventHandler}_{\mathbf{ctx}}(id), \cdots) \\ \mathit{Doc}(id) \in \mathit{Docs}(id) \qquad \mathit{Doc}(id) = (\cdots, \mathit{nodes}(id), \cdots) \\ \mathit{node}(id) \in \mathit{nodes}(id) \qquad \mathit{node}(id) = (id, \mathit{attributes}, \mathit{nodes}_1, \mathit{content}, \| \|, \ell) \\ \mathit{EventHandler}_{\mathbf{ctx}}(id) = (\cdots, \mathsf{some}(R, id_r)) \\ \psi = \mathit{ayncAPIRetState}(R, \ell^-, id_r, v) \qquad \Sigma'_{ctx}(id) = \Sigma_{ctx}(id)[\Psi \Leftarrow \Psi :: \psi]\end{array}}{\Sigma_{ctx} \| \Gamma, \mathsf{ret}\ v \|_{id}; \mathcal{E} \xrightarrow{\tau} \Sigma'_{ctx} \| \Gamma, \mathsf{skip} \|_{id}; \mathcal{E}} \text{ RETURN-PAGE}$$

When the returning event handler is a blocking event handler in the core, a new browser state $\mathsf{DoneBlkEvtState}(\ell^-, id_r, e, v)$ is generated to signal the completion of processing that blocking event.

$$\frac{\begin{array}{l}\Sigma_{ctx}(id) = (\Psi, \mathit{Tabs}, \mathit{ExtCoreRs}_{\mathbf{ctx}}(id), \cdots) \\ \Psi = \Psi' :: \mathsf{ProcBlkEvtState}(\ell^-, id_r, e) \qquad \mathit{ExtCoreR}_{\mathbf{ctx}}(id) \in \mathit{ExtCoreRs}_{\mathbf{ctx}}(id) \\ \mathit{ExtCoreR}_{\mathbf{ctx}}(id) = (id, \_, \mathit{EventHandlers} :: \mathit{EventHandler}_{\mathbf{ctx}}, \ell) \\ \mathit{EventHandler}_{\mathbf{ctx}}(id) = (\cdots, \mathsf{blocking}, \mathsf{some}(e, id_r)) \\ \psi = \mathsf{DoneBlkEvtState}(\ell^-, id_r, e, v) \qquad \Sigma'_{ctx}(id) = \Sigma_{ctx}(id)[\Psi \Leftarrow \Psi :: \psi]\end{array}}{\Sigma_{ctx} \| \Gamma, \mathsf{ret}\ v \|_{id}; \mathcal{E} \xrightarrow{\tau} \Sigma'_{ctx} \| \Gamma, \mathsf{skip} \|_{id}; \mathcal{E}} \text{ RETURN-CORE-BLOCKING}$$

Calls We model two types of API calls: synchronous calls and asynchronous calls. In general, a caller calls an API with an argument vector $\vec{v}$. The label checking for an API call checks whether the caller has permission to call the API. Here, we allow the caller to apply declassification capabilities before the call.

$$\frac{\ell = \mathsf{ctxOfId}(\Sigma_{ctx} \| \Gamma, \mathit{API}_s(\vec{v}) \|_{id}, id) \qquad \ell^- \not\Rightarrow \mathsf{labOf}(\mathit{API}_s)}{\Sigma_{ctx} \| \Gamma, \mathit{API}_s(\vec{v}) \|_{id}; \mathcal{E} \xrightarrow{\tau} \Sigma_{ctx} \| \Gamma, \mathsf{void} \|_{id}; \mathcal{E}} \text{ SKIPAPICALL-S}$$

$$\frac{\begin{array}{l}\ell = \mathsf{ctxOfId}(\Sigma_{ctx} \| \Gamma, \mathit{API}_s(\vec{v}) \|_{id}, id) \\ \qquad \ell^- \Rightarrow \mathsf{labOf}(\mathit{API}_s) \qquad \Sigma_{ctx} \| \Gamma, \mathit{API}_s(\vec{v}) \|_{id} \xrightarrow{\beta}_{\mathit{API}_s} \Sigma'; \mathcal{E}'\end{array}}{\Sigma_{ctx} \| \Gamma, \mathit{API}_s(\vec{v}) \|_{id}; \mathcal{E} \xrightarrow{\beta} \Sigma'; \mathcal{E} :: \mathcal{E}'} \text{ FIREAPICALL-S}$$

$$\frac{\ell = \mathsf{ctxOfId}(\Sigma_{ctx} \| \Gamma, \mathit{API}_a(\vec{v}) \|_{id}, id) \qquad \ell^- \not\Rightarrow \mathsf{labOf}(\mathit{API}_a)}{\Sigma_{ctx} \| \Gamma, \mathit{API}_a(\vec{v}) \|_{id}; \mathcal{E} \xrightarrow{\tau} \Sigma_{ctx} \| \Gamma, \mathsf{skip} \|_{id}; \mathcal{E}} \text{ SKIPAPICALL-A}$$

There are three asynchronous API fire rules; each adds a new event handler to the context and adds to the browser state an indication that the call has fired. This state will be processed using rule INTERNALSTATETRANSITION2 (Section 3.9). When calling an API, the scripts can use declassification capabilities, and therefore the browser state associated with the call has the declassified label. For calls that include callback functions, the callback functions will be added as new event handlers in the system. The event type of the event handler for a callback function should uniquely identify that function. Events triggering these event handlers will be generated once the asynchronous API finishes and returns its return result.

$$\frac{\begin{array}{l}\Sigma_{ctx}(id) = (\Psi, Tabs, ExtCoreRs_{\mathsf{ctx}}(id), \cdots) \qquad ExtCoreR_{\mathsf{ctx}}(id) \in ExtCoreRs_{\mathsf{ctx}}(id) \\ ExtCoreR_{\mathsf{ctx}}\llbracket \Gamma, API_a(\vec{v}) \rrbracket_{id} = (id, \_, EventHandlers, \ell) \qquad \ell^- \sqsubseteq \mathsf{labOf}(API_a) \\ \ell \Rightarrow \kappa_t \\ callbackOf(\vec{v}) = x.cmd \\ fresh(id_{cb}) \qquad fresh(id_h) \qquad eh = (id_h, id_{cb}, x.cmd, \cdot, \mathsf{skip}, \mathsf{void}, \mathsf{nonblocking}, \mathsf{none}) \\ ExtCoreR_{\mathsf{ctx}}{}'(id) = ExtCoreR_{\mathsf{ctx}}(id)[EventHandlers \Leftarrow EventHandlers :: eh] \\ ExtCoreRs_{\mathsf{ctx}}{}'(id) = ExtCoreRs_{\mathsf{ctx}}(id)[ExtCoreRs_{\mathsf{ctx}}(id) \Leftarrow ExtCoreRs_{\mathsf{ctx}}{}'(id)] \\ \psi = ayncAPIstate(\kappa_t, id_{cb}, API_a(\vec{v})) \\ \Sigma'_{ctx}(id) = \Sigma_{ctx}(id)[ExtCoreRs_{\mathsf{ctx}}(id) \Leftarrow ExtCoreRs'_{\mathsf{ctx}}(id)][\Psi \Leftarrow \Psi :: \psi]\end{array}}{\Sigma_{ctx}\llbracket \Gamma, API_a(\vec{v}) \rrbracket_{id}; \mathcal{E} \xrightarrow{\tau} \Sigma'_{ctx}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}; \mathcal{E}} \text{ FIREAPICALL-A-CORE}$$

$$\frac{\begin{array}{l}\Sigma_{ctx}(id) = (\Psi, Tabs_{\mathsf{ctx}}(id), \cdots) \qquad t_{\mathsf{ctx}}(id) \in Tabs_{\mathsf{ctx}}(id) \\ t_{\mathsf{ctx}}(id) = (\cdots, Docs_{\mathsf{ctx}}(id), \cdots) \qquad Doc_{\mathsf{ctx}}(id) \in Docs_{\mathsf{ctx}}(id) \\ Doc_{\mathsf{ctx}}(id) = (\cdots, DocCSs_{\mathsf{ctx}}(id), \cdots) \qquad DocCS_{\mathsf{ctx}}(id) \in DocCSs_{\mathsf{ctx}}(id) \\ DocCS_{\mathsf{ctx}}(id) = (\_, id, \cdots, EventHandlers_{\mathsf{ctx}}(id), \ell) \qquad \ell^- \sqsubseteq \mathsf{labOf}(API_a) \\ \ell \Rightarrow \kappa_t \\ callbackOf(\vec{v}) = x.cmd \\ fresh(id_{cb}) \qquad fresh(id_h) \qquad eh = (id_h, id_{cb}, x.cmd, \cdot, \mathsf{skip}, \mathsf{void}, \mathsf{nonblocking}, \mathsf{none}) \\ DocCS'_{\mathsf{ctx}}(id) = DocCS_{\mathsf{ctx}}(id)[EventHandlers_{\mathsf{ctx}}(id) \Leftarrow EventHandlers_{\mathsf{ctx}}(id) :: eh] \\ DocCSs'_{\mathsf{ctx}}(id) = DocCSs_{\mathsf{ctx}}(id)[DocCS_{\mathsf{ctx}}(id) \Leftarrow DocCS'_{\mathsf{ctx}}(id)] \\ Doc'_{\mathsf{ctx}}(id) = Doc_{\mathsf{ctx}}(id)[DocCS_{\mathsf{ctx}}(id) \Leftarrow DocCS'_{\mathsf{ctx}}(id)] \\ t'_{\mathsf{ctx}}(id) = t_{\mathsf{ctx}}(id)[Doc_{\mathsf{ctx}}(id) \Leftarrow Doc'_{\mathsf{ctx}}(id)] \qquad \psi = ayncAPIstate(\kappa_t, id_{cb}, API_a(\vec{v})) \\ \Sigma'_{ctx}(id) = \Sigma_{ctx}(id)[t_{\mathsf{ctx}}(id) \Leftarrow t'_{\mathsf{ctx}}(id)][\Psi \Leftarrow \Psi :: \psi]\end{array}}{\Sigma_{ctx}\llbracket \Gamma, API_a(\vec{v}) \rrbracket_{id}; \mathcal{E} \xrightarrow{\tau} \Sigma'_{ctx}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}; \mathcal{E}} \text{ FIREAPICALL-A-CS}$$

$$\frac{\begin{array}{l}\Sigma_{ctx}(id) = (\Psi, Tabs_{\mathsf{ctx}}(id), \cdots) \\ t_{\mathsf{ctx}}(id) \in Tabs_{\mathsf{ctx}}(id) \qquad t_{\mathsf{ctx}}(id) = (id_t, Docs(id), url, EventHandlers_{\mathsf{ctx}}(id), \cdots) \\ Doc(id) \in Docs(id) \qquad Doc(id) = (\cdots, nodes(id), \cdots) \qquad node(id) \in nodes(id) \\ node(id) = (id, attributes, nodes_1, content, \ell) \qquad \ell^- \sqsubseteq \mathsf{labOf}(API_a) \\ \kappa_t, \ell \Rightarrow \kappa_t \\ callbackOf(\vec{v}) = x.cmd \\ fresh(id_{cb}) \qquad eh = (id, id_{cb}, x.cmd, \cdot, \mathsf{skip}, \mathsf{void}, \mathsf{nonblocking}, \mathsf{none}) \\ t'_{\mathsf{ctx}}(id) = t_{\mathsf{ctx}}(id)[EventHandlers_{\mathsf{ctx}}(id) \Leftarrow EventHandlers_{\mathsf{ctx}}(id) :: eh] \\ \psi = ayncAPIstate(\kappa_t, id_{cb}, API_a(\vec{v})) \\ \Sigma'_{ctx}(id) = \Sigma_{ctx}(id)[t_{\mathsf{ctx}}(id) \Leftarrow t'_{\mathsf{ctx}}(id)][\Psi \Leftarrow \Psi :: \psi]\end{array}}{\Sigma_{ctx}\llbracket \Gamma, API_a(\vec{v}) \rrbracket_{id}; \mathcal{E} \xrightarrow{\tau} \Sigma'_{ctx}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}; \mathcal{E}} \text{ FIREAPICALL-A-PAGE}$$

3.4 Event Enqueue Rules

Fire nonblocking stateless events For each handler eh of e, the system adds e to the event queue of the handler eh.

If the event is not linked to any browser state, it can be processed regardless of the browser state.

$$\frac{\begin{array}{l} e \text{ is not a blocking event} \qquad \Sigma = (\Psi, \mathit{Tabs}, \mathit{ExtCoreRs}, \cdots) \\ \Sigma' = \Sigma[\mathit{ExtCoreRs} \Leftarrow \mathit{ExtCoreRs} \lhd_Q e][\mathit{Tabs} \Leftarrow \mathit{Tabs} \lhd_Q e] \end{array}}{\Sigma; \mathcal{E} :: e \xrightarrow{\tau} \Sigma'; \mathcal{E}} \textsc{EventFire1}$$

The following rule enqueues a nonblocking event that can only be processed at certain browser states. Further, after going through all the extensions, the browser moves to the next state.

For instance, $\psi = \mathsf{ws.responseStarted}(\kappa, id_t, id_d, id_n, {id_e}', url, \mathit{info}')$ is related to $e = ({id_e}', \mathsf{webRequest.onResponceStarted}, \mathsf{none}, \mathit{info}', \kappa)$ $(\psi \sim e)$, and the next browser state ψ is $\mathsf{ws.completed}(\cdots)$.

$$\frac{\begin{array}{l} e \text{ is not a blocking event} \\ \Sigma = (\Psi, \mathit{Tabs}, \mathit{ExtCoreRs}, \cdots) \qquad \Psi = \Psi' :: \psi \qquad \psi \sim_{nb} e \qquad (\psi', e') = \mathsf{nextState}(\psi, \mathsf{void}) \\ \Sigma' = \Sigma[\Psi \Leftarrow \Psi' :: \psi'][\mathit{ExtCoreRs} \Leftarrow \mathit{ExtCoreRs} \lhd_Q e][\mathit{Tabs} \Leftarrow \mathit{Tabs} \lhd_Q e] \end{array}}{\Sigma; \mathcal{E} :: e \xrightarrow{\tau} \Sigma'; \mathcal{E} :: e'} \textsc{EventFire2}$$

We may also use nextStateC to advance state. In the following rule, besides the event queues of the event handlers, updates other parts of the system as well.

$$\frac{\begin{array}{l} e \text{ is not a blocking event} \qquad \Sigma = (\Psi, \mathit{Tabs}, \mathit{ExtCoreRs}, \cdots) \\ \Psi = \Psi' :: \psi \qquad \psi \sim_{nb} e \qquad \Sigma' = \Sigma[\mathit{ExtCoreRs} \Leftarrow \mathit{ExtCoreRs} \lhd_Q e][\mathit{Tabs} \Leftarrow \mathit{Tabs} \lhd_Q e] \\ (\Sigma'', \mathcal{E}_1) = \mathsf{nextStateC}(\Sigma', \psi) \end{array}}{\Sigma; \mathcal{E} :: e \xrightarrow{\tau} \Sigma''; \mathcal{E} :: \mathcal{E}_1} \textsc{EventFire3}$$

Fire blocking events that have no blocking handlers In Chromium, only extension cores contain blocking event handlers. Given a blocking event, the non-blocking handlers with a label higher than the event's and the blocking handlers with a label the same as the event's can process the event. The browser can only move to the next state after all the blocking handlers have processed the event. If there is no blocking event handler with the same label as the event's, after enqueuing the event to all its handlers, the browser can advance to the next step directly.

$$\frac{\begin{array}{l} e \text{ is a blocking event} \qquad \Sigma = (\Psi, \mathit{Tabs}, \mathit{ExtCoreRs}, \cdots) \qquad \Psi = \Psi' :: \psi \qquad \psi \sim_b e \\ \forall \mathit{ExtCoreR} \in \mathit{ExtCoreRs}, \mathit{EHs} = \mathit{getActiveHandlers}(\mathit{ExtCoreR}, e), \\ \quad \forall eh \in \mathit{EHs}, \mathsf{labOf}(eh)^- = \mathsf{labOf}(e) \Rightarrow \mathit{isBlocking}(eh) = \mathsf{false} \\ (\psi', e') = \mathsf{nextState}(\psi, \mathsf{void}) \qquad \Sigma' = \Sigma[\Psi \Leftarrow \Psi' :: \psi'][\mathit{ExtCoreRs} \Leftarrow \mathit{ExtCoreRs} \lhd_Q e] \end{array}}{\Sigma; \mathcal{E} :: e \xrightarrow{\tau} \Sigma'; \mathcal{E} :: e'} \textsc{EventFireNonBL}$$

Fire a blocking event There must exist at least one blocking handler with the same label as the event's. The browser enqueues the event.

$$\frac{\begin{array}{l} e \text{ is a blocking event} \qquad \Sigma = (\Psi :: \psi, \mathit{Tabs}, \mathit{ExtCoreRs}, \cdots) \qquad \psi \sim_b e \\ \exists \mathit{ExtCoreR} \in \mathit{ExtCoreRs}, \mathit{EHs} = \mathit{getActiveHandlers}(\mathit{ExtCoreR}, e), \\ \quad \exists eh \in \mathit{EHs}, \mathit{isBlocking}(eh) = \mathsf{true} \wedge \mathsf{labOf}(eh)^- = \mathsf{labOf}(e) \\ \Sigma' = \Sigma[\mathit{ExtCoreRs} \Leftarrow \mathit{ExtCoreRs} \lhd_Q e] \end{array}}{\Sigma; \mathcal{E} :: e \xrightarrow{\tau} \Sigma'; \mathcal{E}} \textsc{EventFireBL}$$

Blocking handler finished (1) All the blocking handlers for the event have finished processing the event, and none of them returns a value to change the browser behavior. The browser moves to the next step using nextstate based on the browser state related to the event.

$$\frac{\begin{array}{l} e \text{ is a blocking event} \qquad e \sim_b \psi \qquad \Sigma = (\Psi, \mathit{Tabs}, \mathit{ExtCoreRs}, \cdots) \qquad \Psi = \Psi_1 :: \psi :: \Psi_2 \\ \nexists \psi \in \Psi_1, s.t.\psi = \mathsf{ProcBlkEvtState}(\_, \_, e) \text{ or } \mathsf{DoneBlkEvtState}(\cdots, e, \_) \\ \Psi_2 \text{ contains matching pairs of } \mathsf{ProcBlkEvtState}(\_, id_r, e) \text{ and } \mathsf{DoneBlkEvtState}(\_, id_r, e, 0) \\ \forall \mathit{ExtCoreR} \in \mathit{ExtCoreRs}, \mathit{EHs} = \mathit{getActiveHandlers}(\mathit{ExtCoreR}, e), \\ \quad \forall eh \in \mathit{EHs}, \mathit{labOf}(eh)^{-} = \mathit{labOf}(e) \wedge \mathit{isBlocking}(eh) = \mathsf{true}, e \notin \mathit{Qof}(eh) \\ (\psi', \mathcal{E}_1) = \mathsf{nextState}(\psi, \mathsf{noChange}) \qquad \Sigma' = \Sigma[\Psi \Leftarrow \Psi_1 :: \psi'] \end{array}}{\Sigma; \mathcal{E} \xrightarrow{\tau} \Sigma'; \mathcal{E} :: \mathcal{E}_1} \text{ BLOCKFINISHED1}$$

Blocking handler finished (2) All the blocking handlers for the event have finished processing the event. Every blocking handler with the same label as the event's has returned a value. The system chooses the return value that comes from the last installed extension's handler. Then the browser moves to a new state according to the current browser state and the selected return value. $\mathcal{E}_1$ denotes the subsequent events triggered.

$$\frac{\begin{array}{l} e \text{ is a blocking event} \qquad e \sim_b \psi \qquad \Sigma = (\Psi, \mathit{Tabs}, \mathit{ExtCoreRs}, \cdots) \qquad \Psi = \Psi_1 :: \psi :: \Psi_2 \\ \nexists \psi \in \Psi_1, s.t.\psi = \mathsf{ProcBlkEvtState}(\_, \_, e) \text{ or } \mathsf{DoneBlkEvtState}(\cdots, e, \_) \\ \Psi_2 \text{ contains matching pairs of } \mathsf{ProcBlkEvtState}(\_, id_i, e) \text{ and } \mathsf{DoneBlkEvtState}(\_, id_i, e, c_i) \\ \forall \mathit{ExtCoreR} \in \mathit{ExtCoreRs}, \mathit{EHs} = \mathit{getActiveHandlers}(\mathit{ExtCoreR}, e), \\ \quad \forall eh \in \mathit{EHs}, \mathit{labOf}(eh)^{-} = \mathit{labOf}(e) \wedge \mathit{isBlocking}(eh) = \mathsf{true}, e \notin \mathit{Qof}(eh) \\ \mathit{getLastInstalledReturn}(\Psi_2) = (\kappa, id_j, c_j) \text{ s.t. } c_j \neq 0 \\ (\psi', \mathcal{E}_1) = \mathsf{nextState}(\psi, \mathsf{Blocked}(c_j)) \qquad \Sigma' = \Sigma[\Psi \Leftarrow \Psi_1 :: \psi'] \end{array}}{\Sigma; \mathcal{E} \xrightarrow{\tau} \Sigma'; \mathcal{E} :: \mathcal{E}_1} \text{ BLOCKFINISHED2}$$

3.5 Event Dequeue Rules

Dequeue an event in CS In content scripts, after finishing the processing of an event, a handler dequeues the next event e in its event queue. If e's label is not lower than the hosting content script's label, e will be discarded; else, the event handler starts processing e. As long as e is processed by the handler, the hosting content script label is tainted with e's label.

$$\frac{\begin{array}{l} \Sigma_{ctx}(id) = (\Psi, \mathit{Tabs}_{\mathbf{ctx}}(id), \cdots) \qquad t_{\mathbf{ctx}}(id) \in \mathit{Tabs}_{\mathbf{ctx}}(id) \\ t_{\mathbf{ctx}}(id) = (\cdots, \mathit{Docs}_{\mathbf{ctx}}(id), \cdots) \qquad \mathit{Doc}_{\mathbf{ctx}}(id) \in \mathit{Docs}_{\mathbf{ctx}}(id) \\ \mathit{Doc}_{\mathbf{ctx}}(id) = (\cdots, \mathit{DocCSs}_{\mathbf{ctx}}(id), \cdots) \qquad \mathit{DocCS}_{\mathbf{ctx}}(id) \in \mathit{DocCSs}_{\mathbf{ctx}}(id) \\ \mathit{DocCS}_{\mathbf{ctx}}(id) = (\_, id, \_, \llbracket\ \rrbracket, \mathit{EventHandlers} :: eh_{\mathbf{ctx}}, \ell) \\ eh_{\mathbf{ctx}} = (\cdot, \mathit{eventType}, x.cmd, e :: \mathcal{E}_1, \llbracket\rrbracket, id_e, \mathit{BlockingFlag}, \mathit{return}) \\ \mathit{labOf}(e) \not\sqsubseteq \ell^{*} \qquad eh'_{\mathbf{ctx}} = (\cdot, \mathit{eventType}, x.cmd, \mathcal{E}_1, \llbracket\rrbracket, id_e, \mathit{BlockingFlag}, \mathit{return}) \\ \mathit{DocCS}'_{\mathbf{ctx}}(id) = \mathit{DocCS}_{\mathbf{ctx}}(id)[eh_{\mathbf{ctx}} \Leftarrow eh'_{\mathbf{ctx}}] \\ \mathit{DocCSs}'_{\mathbf{ctx}}(id) = \mathit{DocCSs}_{\mathbf{ctx}}(id)[\mathit{DocCS}_{\mathbf{ctx}}(id) \Leftarrow \mathit{DocCS}'_{\mathbf{ctx}}(id)] \\ \mathit{Doc}'_{\mathbf{ctx}}(id) = \mathit{Doc}_{\mathbf{ctx}}(id)[\mathit{DocCSs}_{\mathbf{ctx}}(id) \Leftarrow \mathit{DocCSs}'_{\mathbf{ctx}}(id)] \\ t'_{\mathbf{ctx}}(id) = t_{\mathbf{ctx}}(id)[\mathit{Doc}_{\mathbf{ctx}}(id) \Leftarrow \mathit{Doc}'_{\mathbf{ctx}}(id)] \\ \Sigma'_{ctx}(id) = \Sigma_{ctx}(id)[t_{\mathbf{ctx}}(id) \Leftarrow t'_{\mathbf{ctx}}(id)] \end{array}}{\Sigma_{ctx} \llbracket\, \Gamma, \mathsf{skip} \,\rrbracket_{id}; \mathcal{E} \xrightarrow{\tau} \Sigma'_{ctx} \llbracket\, \Gamma, \mathsf{skip} \,\rrbracket_{id}; \mathcal{E}} \text{ EVENTDEQUEUECS1}$$

$$\frac{\begin{array}{l}
\Sigma_{ctx}(id) = (\Psi, \mathit{Tabs}_{\mathbf{ctx}}(id), \cdots) \qquad t_{\mathbf{ctx}}(id) \in \mathit{Tabs}_{\mathbf{ctx}}(id) \\
t_{\mathbf{ctx}}(id) = (\cdots, \mathit{Docs}_{\mathbf{ctx}}(id), \cdots) \qquad \mathit{Doc}_{\mathbf{ctx}}(id) \in \mathit{Docs}_{\mathbf{ctx}}(id) \\
\mathit{Doc}_{\mathbf{ctx}}(id) = (\cdots, \mathit{DocCSs}_{\mathbf{ctx}}(id), \cdots) \qquad \mathit{DocCS}_{\mathbf{ctx}}(id) \in \mathit{DocCSs}_{\mathbf{ctx}}(id) \\
\mathit{DocCS}_{\mathbf{ctx}}(id) = (\_, id, \_, \llbracket\ \rrbracket, \_, \mathit{EventHandlers} :: \mathit{eh}_{\mathbf{ctx}}, \ell) \\
\mathit{eh}_{\mathbf{ctx}} = (\cdot, \mathit{eventType}, x.\mathit{cmd}, e :: \mathcal{E}_1, \llbracket\rrbracket, id_e, \mathit{BlockingFlag}, \mathit{return}') \\
e = (id'_e, \mathit{eventType}, \mathit{return}, \cdots, \kappa_e) \qquad \kappa_e \sqsubseteq \ell^* \\
\ell' = \kappa_e \rhd_{tnt} \ell \qquad \mathit{eh}' = (\cdot, \mathit{eventType}, x.\mathit{cmd}, \mathcal{E}_1, \llbracket\rrbracket, id'_e, \mathit{BlockingFlag}, \mathit{return}) \\
\mathit{DocCS}'_{\mathbf{ctx}}(id) = \mathit{DocCS}_{\mathbf{ctx}}(id)[\mathit{eh}_{\mathbf{ctx}} \Leftarrow \mathit{eh}'_{\mathbf{ctx}}][\ell \Leftarrow \ell'] \\
\mathit{DocCSs}'_{\mathbf{ctx}}(id) = \mathit{DocCSs}_{\mathbf{ctx}}(id)[\mathit{DocCS}_{\mathbf{ctx}}(id) \Leftarrow \mathit{DocCS}'_{\mathbf{ctx}}(id)] \\
\mathit{Doc}'_{\mathbf{ctx}}(id) = \mathit{Doc}_{\mathbf{ctx}}(id)[\mathit{DocCS}_{\mathbf{ctx}}(id) \Leftarrow \mathit{DocCS}'_{\mathbf{ctx}}(id)] \\
t'_{\mathbf{ctx}}(id) = t_{\mathbf{ctx}}(id)[\mathit{Doc}_{\mathbf{ctx}}(id) \Leftarrow \mathit{Doc}'_{\mathbf{ctx}}(id)] \\
\Sigma'_{ctx}(id) = \Sigma_{ctx}(id)[t_{\mathbf{ctx}}(id) \Leftarrow t'_{\mathbf{ctx}}(id)]
\end{array}}{\Sigma_{ctx} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id}; \mathcal{E} \xrightarrow{\mathit{updLab}(e,\ell'^-)} \Sigma'_{ctx} \llbracket \Gamma, \mathit{cmd}[e/x] \rrbracket_{id}; \mathcal{E}}\ \textsc{EventDequeueCS2}$$

Dequeue an event in Core In extension cores, after finishing processing of an event, a handler dequeues the next event e in its event queue. A non-blocking event handler can process an event if the event's label is lower than the handler's hosting extension core's label. A blocking event handler can process an event and generate a new browser state $\mathsf{ProcBlkEvtState}(\kappa_e, id_r, e)$ if its label (after necessary floating) is the same as the event's, where κ_e is e's label and id_r is a fresh random string. As long as e is processed by the handler, the hosting extension core's label is tainted with e's label.

If a handler cannot process an event due to label checking, the event will be discarded.

$$\frac{\begin{array}{l}
\Sigma_{ctx}(id) = (\Psi, \mathit{Tabs}, \mathit{ExtCoreRs}_{\mathbf{ctx}}(id), \cdots) \qquad \mathit{ExtCoreR}_{\mathbf{ctx}}(id) \in \mathit{ExtCoreRs}_{\mathbf{ctx}}(id) \\
\mathit{ExtCoreR}_{\mathbf{ctx}}(id) = (id, (\llbracket\rrbracket, \_, \mathit{EventHandlers} :: \mathit{eh}_{\mathbf{ctx}}), \ell) \\
\mathit{eh}_{\mathbf{ctx}} = (\cdot, \mathit{eventType}, x.\mathit{cmd}, e :: \mathcal{E}_1, \llbracket\rrbracket, id_e, \mathsf{nonblocking}, \mathit{return}) \\
\mathit{labOf}(e) \not\sqsubseteq \ell^* \qquad \mathit{eh}'_{\mathbf{ctx}} = (\cdot, \mathit{eventType}, x.\mathit{cmd}, \mathcal{E}_1, \llbracket\rrbracket, id_e, \mathsf{nonblocking}, \mathit{return}) \\
\mathit{ExtCoreR}_{\mathbf{ctx}}{}'(id) = \mathit{ExtCoreR}_{\mathbf{ctx}}(id)[\mathit{eh}_{\mathbf{ctx}} \Leftarrow \mathit{eh}'_{\mathbf{ctx}}] \\
\mathit{ExtCoreRs}_{\mathbf{ctx}}{}'(id) = \mathit{ExtCoreRs}_{\mathbf{ctx}}(id)[\mathit{ExtCoreRs}_{\mathbf{ctx}}(id) \Leftarrow \mathit{ExtCoreRs}_{\mathbf{ctx}}{}'(id)] \\
\Sigma'_{ctx}(id) = \Sigma_{ctx}(id)[\mathit{ExtCoreRs}_{\mathbf{ctx}}(id) \Leftarrow \mathit{ExtCoreRs}'_{\mathbf{ctx}}(id)]
\end{array}}{\Sigma_{ctx} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id}; \mathcal{E} \xrightarrow{\tau} \Sigma'_{ctx} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id}; \mathcal{E}}\ \textsc{EventDequeueCore1}$$

$$\frac{\begin{array}{l}
\Sigma_{ctx}(id) = (\Psi, \mathit{Tabs}, \mathit{ExtCoreRs}_{\mathbf{ctx}}(id), \_, \mathit{Exts} :: \mathit{Ext}, \cdots) \\
\mathit{ExtCoreR}_{\mathbf{ctx}}(id) \in \mathit{ExtCoreRs}_{\mathbf{ctx}}(id) \\
\mathit{ExtCoreR}_{\mathbf{ctx}}(id) = (id, (\llbracket\rrbracket, \_, \mathit{EventHandlers} :: \mathit{eh}_{\mathbf{ctx}}), \ell) \\
\mathit{eh}_{\mathbf{ctx}} = (\cdot, \mathit{eventType}, x.\mathit{cmd}, e :: \mathcal{E}_1, \llbracket\rrbracket, id_e, \mathsf{nonblocking}, \mathit{return}) \\
e = (id'_e, \mathit{eventType}, \mathit{return}', \cdots, \kappa_e) \qquad \kappa_e \sqsubseteq \ell^* \qquad \ell' = \kappa_e \rhd_{tnt} \ell \\
\mathit{Ext} = (id, \cdots, \ell_s) \qquad \mathit{eh}'_{\mathbf{ctx}} = (\cdot, \mathit{eventType}, x.\mathit{cmd}, \mathcal{E}_1, \llbracket\rrbracket, id'_e, \mathsf{nonblocking}, \mathit{return}') \\
\mathit{ExtCoreR}_{\mathbf{ctx}}{}'(id) = \mathit{ExtCoreR}_{\mathbf{ctx}}(id)[\mathit{eh}_{\mathbf{ctx}} \Leftarrow \mathit{eh}'_{\mathbf{ctx}}][\ell \Leftarrow \ell'] \\
\mathit{ExtCoreRs}_{\mathbf{ctx}}{}'(id) = \mathit{ExtCoreRs}_{\mathbf{ctx}}(id)[\mathit{ExtCoreRs}_{\mathbf{ctx}}(id) \Leftarrow \mathit{ExtCoreRs}_{\mathbf{ctx}}{}'(id)] \\
\mathit{Ext}' = \mathit{Ext}[\ell_s \Leftarrow (\mathit{labOf}(e) \rhd_{tnt} \ell_s)] \\
\Sigma'_{ctx}(id) = \Sigma_{ctx}(id)[\mathit{ExtCoreRs}_{\mathbf{ctx}}(id) \Leftarrow \mathit{ExtCoreRs}'_{\mathbf{ctx}}(id)][\mathit{Ext} \Leftarrow \mathit{Ext}']
\end{array}}{\Sigma_{ctx} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id}; \mathcal{E} \xrightarrow{\mathit{updLab}(e,\ell'^-)} \Sigma'_{ctx} \llbracket \Gamma, \mathit{cmd}[e/x] \rrbracket_{id}; \mathcal{E}}\ \textsc{EventDequeueCore2}$$

$$\frac{\begin{array}{l}\Sigma_{ctx}(id) = (\Psi, \mathit{Tabs}, \mathit{ExtCoreRs}_{\mathsf{ctx}}(id), \cdots) \\ \mathit{ExtCoreR}_{\mathsf{ctx}}(id) \in \mathit{ExtCoreRs}_{\mathsf{ctx}}(id) \\ \mathit{ExtCoreR}_{\mathsf{ctx}}(id) = (id, (\|\|, \_, \mathit{EventHandlers} :: \mathit{eh}_{\mathsf{ctx}}), \ell) \\ \mathit{eh}_{\mathsf{ctx}} = (\cdot, \mathit{eventType}, x.\mathit{cmd}, e :: \mathcal{E}_1, \|\|, id_e, \mathsf{blocking}, \mathit{return}) \\ \mathit{labOf}(e) \neq \ell^- \qquad \mathit{eh}'_{\mathsf{ctx}} = (\cdot, \mathit{eventType}, x.\mathit{cmd}, \mathcal{E}_1, \|\|, id_e, \mathsf{blocking}, \mathit{return}) \\ \mathit{ExtCoreR}_{\mathsf{ctx}}{}'(id) = \mathit{ExtCoreR}_{\mathsf{ctx}}(id)[\mathit{eh}_{\mathsf{ctx}} \Leftarrow \mathit{eh}'_{\mathsf{ctx}}] \\ \mathit{ExtCoreRs}_{\mathsf{ctx}}{}'(id) = \mathit{ExtCoreRs}_{\mathsf{ctx}}(id)[\mathit{ExtCoreRs}_{\mathsf{ctx}}(id) \Leftarrow \mathit{ExtCoreRs}_{\mathsf{ctx}}{}'(id)] \\ \Sigma'_{ctx}(id) = \Sigma_{ctx}(id)[\mathit{ExtCoreRs}_{\mathsf{ctx}}(id) \Leftarrow \mathit{ExtCoreRs}'_{\mathsf{ctx}}(id)]\end{array}}{\Sigma_{ctx} \| \Gamma, \mathsf{skip} \|_{id}; \mathcal{E} \xrightarrow{\tau} \Sigma'_{ctx} \| \Gamma, \mathsf{skip} \|_{id}; \mathcal{E}} \text{ EVENTDEQUEUECOREBLOCKING1}$$

$$\frac{\begin{array}{l}\Sigma_{ctx}(id) = (\Psi, \mathit{Tabs}, \mathit{ExtCoreRs}_{\mathsf{ctx}}(id), \cdots) \\ \mathit{ExtCoreR}_{\mathsf{ctx}}(id) \in \mathit{ExtCoreRs}_{\mathsf{ctx}}(id) \\ \mathit{ExtCoreR}_{\mathsf{ctx}}(id) = (id, (\|\|, \_, \mathit{EventHandlers} :: \mathit{eh}_{\mathsf{ctx}}), \ell) \\ \mathit{eh}_{\mathsf{ctx}} = (\cdot, \mathit{eventType}, x.\mathit{cmd}, e :: \mathcal{E}_1, \|\|, id_e, \mathsf{blocking}, \mathit{return}) \\ e = (id'_e, \mathit{eventType}, \mathit{return}, \cdots, \kappa_e) \\ \kappa_e{=}\ell^- \qquad \mathit{fresh}(id_r) \qquad \psi = \mathsf{ProcBlkEvtState}(\kappa_e, id_r, e) \\ \mathit{eh}'_{\mathsf{ctx}} = (\cdot, \mathit{eventType}, x.\mathit{cmd}, \mathcal{E}_1, \|\|, id'_e, \mathsf{blocking}, \mathsf{some}(e, id_r)) \\ \mathit{ExtCoreR}_{\mathsf{ctx}}{}'(id) = \mathit{ExtCoreR}_{\mathsf{ctx}}(id)[\mathit{eh}_{\mathsf{ctx}} \Leftarrow \mathit{eh}'_{\mathsf{ctx}}] \\ \mathit{ExtCoreRs}_{\mathsf{ctx}}{}'(id) = \mathit{ExtCoreRs}_{\mathsf{ctx}}(id)[\mathit{ExtCoreRs}_{\mathsf{ctx}}(id) \Leftarrow \mathit{ExtCoreRs}_{\mathsf{ctx}}{}'(id)] \\ \Sigma'_{ctx}(id) = \Sigma_{ctx}(id)[\mathit{ExtCoreRs}_{\mathsf{ctx}}(id) \Leftarrow \mathit{ExtCoreRs}'_{\mathsf{ctx}}(id)][\Psi \Leftarrow \Psi :: \psi]\end{array}}{\Sigma_{ctx} \| \Gamma, \mathsf{skip} \|_{id}; \mathcal{E} \longrightarrow \Sigma'_{ctx} \| \Gamma, \mathit{cmd}[e/x] \|_{id}; \mathcal{E}} \text{ EVENTDEQUEUECOREBLOCKING2}$$

Dequeue a page event In page scripts, after finishing processing of an event, a handler dequeues the next event e in its event queue. There are two types of script handlers: inline handlers and the handlers added using "addEventListener". Here we model the second type of handlers. If e's label is not lower than the hosting page script node's label, e will be discarded; else, the event handler starts processing e. As long as e is processed by the handler, the hosting page script node's label is tainted with e's label.

$$\frac{\begin{array}{l}\Sigma_{ctx}(id) = (\Psi, \mathit{Tabs}_{\mathsf{ctx}}(id), \cdots) \qquad t_{\mathsf{ctx}}(id) \in \mathit{Tabs}_{\mathsf{ctx}}(id) \\ t_{\mathsf{ctx}}(id) = (id_t, \mathit{Docs}_{\mathsf{env}}(id), \mathit{url}, \mathit{EventHandlers} :: \mathit{eh}_{\mathsf{ctx}}(id), \_) \\ \mathit{eh}_{\mathsf{ctx}}(id) = (id, \mathit{eventType}, x.\mathit{cmd}, e :: \mathcal{E}_1, \|\|, id_e, \mathit{BlockingFlag}, \mathit{return}) \\ \mathit{Doc}_{\mathsf{env}}(id) \in \mathit{Docs}_{\mathsf{env}}(id) \qquad \mathit{Doc}_{\mathsf{env}}(id) = (\cdots, \mathit{nodes}_{\mathsf{env}}(id), \cdots) \\ \mathit{node}_{\mathsf{env}}(id) \in \mathit{nodes}_{\mathsf{env}}(id) \qquad \mathit{node}_{\mathsf{env}}(id) = (id, \_, \_, \_, \|\|, \ell) \qquad \mathit{labOf}(e) \not\sqsubseteq \ell^* \\ t'_{\mathsf{ctx}}(id) = t_{\mathsf{ctx}}(id)[\mathit{eh}_{\mathsf{ctx}}(id) \Leftarrow \mathit{eh}'_{\mathsf{ctx}}(id)] \qquad \Sigma'_{ctx}(id) = \Sigma_{ctx}(id)[t_{\mathsf{ctx}}(id) \Leftarrow t'_{\mathsf{ctx}}(id)]\end{array}}{\Sigma_{ctx} \| \Gamma, \mathsf{skip} \|_{id}; \mathcal{E} \xrightarrow{\tau} \Sigma'_{ctx} \| \Gamma, \mathsf{skip} \|_{id}; \mathcal{E}} \text{ EVENTDEQUEUEPAGE1}$$

$$\frac{\begin{array}{l}\Sigma_{ctx}(id) = (\Psi, \mathit{Tabs}_{\mathit{ctx}}(id), \cdots) \qquad t_{\mathit{ctx}}(id) \in \mathit{Tabs}_{\mathit{ctx}}(id)\\ t_{\mathit{ctx}}(id) = (id_t, \mathit{Docs}_{\mathit{env}}(id), url, \mathit{EventHandlers} :: eh_{\mathit{ctx}}(id), \mathcal{E}_1 :: e, \_)\\ \mathit{Doc}_{\mathit{env}}(id) \in \mathit{Docs}_{\mathit{env}}(id) \qquad \mathit{Doc}_{\mathit{env}}(id) = (\cdots, \mathit{nodes}_{\mathit{env}}(id), \cdots)\\ \mathit{node}_{\mathit{env}}(id) \in \mathit{nodes}_{\mathit{env}}(id) \qquad \mathit{node}_{\mathit{env}}(id) = (id, \mathit{attributes}, \mathit{nodes}_1, \mathit{content}, \llbracket\rrbracket, \ell)\\ eh_{\mathit{ctx}} = (id, \mathit{eventType}, x.\mathit{cmd}, e :: \mathcal{E}_1, \llbracket\rrbracket, id_e, \mathit{BlockingFlag}, \mathit{return}')\\ e = (id'_e, \mathit{eventType}, \mathit{return}, \cdots, \kappa_e) \qquad \kappa_e \sqsubseteq \ell^*\\ \ell' = \kappa_e \rhd_{tnt} \ell \qquad eh'_{\mathit{ctx}} = (id, \mathit{eventType}, x.\mathit{cmd}, \mathcal{E}_1, \llbracket\rrbracket, id'_e, \mathit{BlockingFlag}, \mathit{return})\\ \mathit{node}'_{\mathit{env}}(id) = (id, \mathit{attributes}, \mathit{nodes}_1, \mathit{content}, \llbracket\rrbracket, \ell')\\ \mathit{Docs}'_{\mathit{env}}(id) = \mathit{Docs}_{\mathit{env}}(id)[\mathit{node}_{\mathit{env}}(id) \Leftarrow \mathit{node}'_{\mathit{env}}(id)]\\ t'_{\mathit{ctx}}(id) = t_{\mathit{ctx}}(id)[\mathit{Doc}_{\mathit{ctx}}(id) \Leftarrow \mathit{Doc}'_{\mathit{ctx}}(id)]\\ \Sigma'_{ctx}(id) = \Sigma_{ctx}(id)[t_{\mathit{ctx}}(id) \Leftarrow t'_{\mathit{ctx}}(id)]\end{array}}{\Sigma_{ctx} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id}; \mathcal{E} \overset{\mathit{updLab}(e,\ell'^-)}{\longrightarrow} \Sigma'_{ctx} \llbracket \Gamma, \mathit{cmd}[e/x] \rrbracket_{id}; \mathcal{E}} \text{ EVENTDEQUEUEPAGE2}$$

3.6 Special Event Processing Rules

Sending data The browser sends a web request to the server and the request is accepted by the server.

$$\frac{e = (\_, \mathsf{sendToSever}, \_, (url, id_e), \kappa)}{\Xi :: (url, \mathsf{listen}; x.\exp); \Sigma; \mathcal{E} :: e \overset{\mathsf{sendToSever}(url, r, \kappa)}{\longrightarrow} \Xi :: (url, \exp[r/x]); \Sigma; \mathcal{E}} \text{ HEADERRECEIVEDBYSERVER}$$

Receiving data The web server sends header information to the browser. When the header arrives at the browser, the browser generates an internal browser state so the data is processed later.

$$\frac{\mathit{fresh}(id) \qquad \kappa = \mathit{labFrom}(url) \qquad e = (id, \mathsf{headerSent}, \mathsf{none}, \mathit{info}, \kappa)}{\Xi :: (url, \mathsf{sendHeader}(h); \exp); \Sigma; \mathcal{E} \overset{\mathsf{serverSndHeader}(h,\kappa)}{\longrightarrow} \Xi :: (url, \exp); \Sigma; \mathcal{E} :: e} \text{ SERVERSNDHDR}$$

The browser has sent a web request header to a server; it has a browser state $\mathsf{ws.headerSent}(\kappa, id_{cb}, id_e, url, \mathit{info})$ and is waiting for the header from the server. Upon receiving a $\mathsf{headerSent}$ event e, if e and $\mathsf{ws.headerSent}(\kappa, id_{cb}, id_e, url, \mathit{info})$ are paired (through info), the browser steps from the $\mathsf{ws.headerSent}(\cdots)$ state to a $\mathsf{headerRecved}(\cdots)$ state, and issues a $\mathsf{webRequest.onHeadersReceived}$ event.

We know that $\kappa \sqsubseteq \mathit{labFrom}(url)$ as we checked labels when the web request was initialized. And by default, $\kappa' = \mathit{labFrom}(url)$ (see rule SERVERSNDHDR).

$$\frac{\begin{array}{l}\Sigma = (\Psi, \cdots) \qquad \Psi = \Psi_1 :: \mathsf{ws.headerSent}(\kappa, id_{cb}, id_e, url, \mathit{info})\\ e = (id, \mathsf{headerSent}, \mathsf{none}, \mathit{info}, \kappa') \qquad \psi = \mathsf{headerRecved}(url, id_{cb}, \kappa \uplus_M \kappa')\\ \mathit{fresh}(id_e) \qquad e' = (id_e, \mathsf{webRequest.onHeadersReceived}, \mathsf{none}, \mathit{info}, \kappa)\end{array}}{\Sigma; \mathcal{E} :: e \overset{\tau}{\longrightarrow} \Sigma[\Psi \Leftarrow \Psi_1 :: \psi]; \mathcal{E} :: e'} \text{ HEADERRECVD}$$

The server sends content to the browser.

$$\frac{\mathit{fresh}(id) \qquad \kappa = \mathit{labFrom}(url) \qquad e = (id, \mathsf{contentSent}, \mathsf{none}, \mathit{info}, \kappa)}{\Xi :: (url, \mathsf{sendContent}(h); \exp); \Sigma; \mathcal{E} \overset{\mathsf{serverSndContent}(c,\kappa)}{\longrightarrow} \Xi :: (url, \exp); \Sigma; \mathcal{E} :: e} \text{ SERVERSNDCONT}$$

The browser receives content and generates an internal state to indicate that the content needs to be processed later.

$$\frac{\Sigma = (\Psi, \cdots) \qquad e = (id, \mathsf{contentSent}, \mathsf{none}, \mathit{info}, \kappa) \qquad \psi = \mathsf{contentRecved}(url, \mathit{info}, \kappa)}{\Sigma; \mathcal{E} :: e \xrightarrow{\tau} \Sigma[\Psi \Leftarrow \Psi :: \psi]; \mathcal{E}} \ \textsc{contentRecvd}$$

3.7 Browser State Transitions

The sequence of transitions starts when a web request is made, either because of an update to a tab's URL; or due to requests made to obtain external resources during DOM tree construction; or because of a DOM write that updates a node and requires contacting remote servers.

3.7.1 Deciding Whether to Send webRequest

Here the browser is going to fetch an object such as a script or an image for a DOM node. The node is assigned with the external object' URL. The node and a readyToFetch state were created during the DOM tree creation procedure (see CREATEDOC(2)) or added by a script (DOMAPPENDCHILDREN(1)). The readyToFetch state has several arugments and has the form: $\mathsf{readyToFetch}(\kappa, (id_t, id_d, id_n), url_{\mathit{res}})$, where κ is the simplified node label, (id_t, id_d, id_n) indicates which node is going to load the object, and url_{res} is the external object's URL. If the resource fetching is allowed, the browser implicitly launches a web request call and generates a $\mathsf{ws.beforeRequest}(\kappa, id_{cb}, (id_t, id_d, id_n), id_e, \mathit{info}, url)$ state.

$$\frac{\begin{array}{l}\Sigma = (\Psi :: \psi, \mathit{Tabs} :: t, \cdots) \qquad \psi = \mathsf{readyToFetch}(\kappa, (id_t, id_d, id_n), url) \\ t = (id_t, \mathit{Docs} :: \mathit{Doc}, url_1, \_, \ell_t) \qquad \mathit{Doc} = (id_d, url, \mathit{nodes}, \mathit{DocCSs}, \chi, \ell_d) \\ \mathit{node} \in_b \mathit{nodes} \qquad \mathit{node} = (id_n, \cdots, \ell_n) \qquad \mathit{NetDeclassify}(\ell_n{}^-) \sqsubseteq \mathit{labFrom}(url) \\ \psi' = \mathsf{webRequestGet}(\kappa, \cdot, (id_t, id_d, id_n), \mathit{info}, url) \qquad \Sigma' = \Sigma[\psi \Leftarrow \psi']\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{readyToFetch}(\kappa, id_t, id_d, id_p, url)) = \Sigma'; \cdot} \ \textsc{fetchAllowed}$$

The object is not allowed to be loaded in a node. We still keep the node, because it may have sub-nodes.

$$\frac{\begin{array}{l}\Sigma = (\Psi :: \psi, \mathit{Tabs} :: t, \cdots) \\ \psi = \mathsf{readyToFetch}(\kappa, (id_t, id_d, id_n), url_{\mathit{res}}) \qquad t = (id_t, \mathit{Docs} :: \mathit{Doc}, url_1, \_, \ell_t) \\ \mathit{Doc} = (id_d, url, \mathit{nodes}, \mathit{DocCSs}, \chi, \ell_d) \qquad \mathit{node} \in_b \mathit{nodes} \\ \mathit{node} = (id_n, \cdots, \ell_n) \qquad \mathit{NetDeclassify}(\ell_n{}^-) \not\sqsubseteq \mathit{labFrom}(url_{\mathit{res}}) \qquad \Sigma' = \Sigma[\Psi :: \psi \Leftarrow \Psi]\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{readyToFetch}(\kappa, id_t, id_d, id_p, url_{\mathit{res}})) = \Sigma'; e = \cdot} \ \textsc{fetchBlocked}$$

Now the system has a readyToFetchDoc state with arguments $(\kappa, id_t, \cdot, \cdot, url)$. It means the browser is going to load a new top-level document to a tab. ℓ_t is the tab's label. κ is the readyToFetchDoc state's label. url is the top-level document's URL. Given ℓ_t, κ, and url, $\mathit{computeLabel}$ works as follows: if $\ell_t \Rightarrow \mathit{labFrom}(url)$, it outputs $\ell_t \uplus_M \mathit{labFrom}(url)$; else, it outputs $\emptyset$. We don't check whether $\kappa \sqsubseteq \mathit{labOf}(url)$, as κ comes from ℓ_t, and κ could be not lower than $\mathit{labFrom}(url)^-$.

If $\mathit{computeLabel}$'s output is not equal to $\emptyset$, then the system will allow the top-level document to be loaded to the tab. A new blank document will be created and added to the tab. The system will transfer from the readytoCreateDoc state to a ws.beforeRequest state.

$$\frac{\begin{array}{l}\Sigma = (\Psi :: \psi, Tabs :: t, \cdots) \qquad t = (id_t, Docs, url_t, \cdots, \ell_t) \\ \psi = \mathsf{readyToFetchDoc}(\kappa, id_t, \cdot, \cdot, url) \qquad \ell_d = computeLabel(\ell_t, \kappa, url) \\ \ell_d \neq \emptyset \qquad fresh(id_d) \qquad Doc_{new} = (id_d, \mathsf{url}, \cdot, \cdot, \cdot, \ell_d) \qquad Docs' = Docs :: Doc_{new} \\ \text{“}doc\text{''} \in info \qquad \psi' = \mathsf{webRequestGet}(\kappa, \cdot, (id_t, \cdot, \cdot), info, url) \\ t' = t[Docs \Leftarrow Docs'][url_t \Leftarrow url] \qquad \Sigma' = \Sigma[t \Leftarrow t'][\psi \Leftarrow \psi']\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{readyToFetchDoc}(\kappa, id_t, \cdot, \cdot, url) = \Sigma'; \cdot} \textsc{FetchDocAllowedTop}$$

If the output of *computeLabel* is an empty set, the top-level document cannot be loaded. The system then discards the readyToFetchDoc state.

$$\frac{\begin{array}{l}\Sigma = (\Psi :: \psi, Tabs :: t, \cdots) \qquad t = (id_t, Docs, url_t, \cdots, \ell_t) \\ \psi = \mathsf{readyToFetchDoc}(\kappa, id_t, \cdot, \cdot, url) \\ computeLabel(\ell_t, \kappa, url) = \emptyset \qquad \Sigma' = \Sigma[\psi \Leftarrow \cdot]\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{readyToFetchDoc}(\kappa, id_t, \cdot, \cdot, url) = \Sigma'; \cdot} \textsc{FetchDocBlockedTop}$$

The to-be-fetched doc is an iframed sub-doc. We need to consider the policy composition problem when generating the new doc's label. As defined in Section 1.3, the function *computeFrameLab* takes four arguments, the iframe's label, the embedded page's label, the generalized CSP, and the url of the framed page. Note that the iframe is a node of its parent doc, so the iframe's label is the node's label. The embedded page's label comes directly from its URL. If the output of *computeFrameLab* is not an empty set, then the fetching of the sub-doc is allowed. Here we do not check whether $\kappa \sqsubseteq labFrom(url_{framed})$ as κ comes from $labOf(id_n)$.

$$\frac{\begin{array}{l}\Sigma = (\Psi :: \psi, Tabs :: t, \cdots) \qquad t = (id_t, Docs, \cdots) \\ \psi = \mathsf{readyToFetchDoc}(\kappa, id_t, id_p, id_n, url_{framed}) \qquad Doc \in Docs \\ Doc = (\_, \_, nodes, \_, \chi, \_) \qquad node \in nodes \qquad node = (id_n, \cdots, \ell_n) \\ \ell_d = computeFrameLab(\ell_n, labFrom(url_{framed}), \chi, url_{framed}) \qquad \ell_d \neq \emptyset \\ fresh(id_d) \qquad Doc_{new} = (id_d, url_{framed}, \cdot, \cdot, \cdot, \ell_d) \qquad Docs' = Docs :: Doc_{new} \\ fresh(id_e) \qquad \psi' = \mathsf{webRequestGet}(\kappa, \cdot, (id_t, id_d, id_n), id_e, info_1, url_{framed}) \\ \text{“}Doc\text{''} \in info_1 \qquad t' = t[Docs \Leftarrow Docs'] \qquad \Sigma' = \Sigma[t \Leftarrow t'][\psi \Leftarrow \psi']\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{readyToFetchDoc}(\kappa, id_t, id_p, id_n, url_{framed}) = \Sigma'; \cdot} \textsc{FetchDocAllowedSub}$$

The output of *computeFrameLab* is an empty set, then the fetching of the sub-doc is blocked.

$$\frac{\begin{array}{l}\Sigma = (\Psi :: \psi, Tabs :: t, \cdots) \\ \psi = \mathsf{readyToFetchDoc}(\kappa, id_t, id_p, id_n, url_{framed}) \\ t = (id_t, Docs, \cdots) \qquad Doc \in Docs \\ Doc = (\_, \_, nodes, \_, \chi, \_) \qquad node \in nodes \qquad node = (id_n, \cdots, \ell_n) \\ computeFrameLab(\ell_n, labFrom(url_{framed}), \chi, url_{framed}) = \emptyset \qquad \Sigma' = \Sigma[\psi \Leftarrow \cdot]\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{readyToFetchDoc}(\kappa, id_t, id_p, id_n, url_{framed}) = \Sigma'; \cdot} \textsc{FetchDocBlockedSub}$$

3.7.2 Web State Transitions

For blocking handlers, we will restrict that only the handler/core with the same label as the event/state can return a value. So we don't need to check the label in nextState.

For web requests for loading external objects/pages, we check whether they can go out according to the rules in Section 3.7.1. For web requests issued by calling HTTP WebRequest APIs, we check whether they can go out in the HTTPWebRequest asyAPICall rules. So given a web request related browser state ws. $*\,(\kappa, \cdots, url, \_\,)$ from an entity with label ℓ, by default, we have $\ell \Rightarrow labFrom(url)^{-}$. However, it is not necessary to require that $\kappa \sqsubseteq labFrom(url)^{-}$.

Before the request is sent $\mathsf{ws.beforeRequest}(\cdots)$ is a browser state related to a blocking event $\mathsf{webRequest.onBeforeRequest}(\cdots)$. $updates$ is the selected output from all the blocking handlers for the event. The browser decides which is the next step based on $updates$.

$$\frac{\begin{array}{l}\Sigma = (\Psi :: \psi, \cdots) \qquad \psi' = \mathsf{ws.beforeRequest}(\kappa, \cdot, (id_t, id_d, id_n), id_e, info, url_{res}) \\ fresh(id_e) \qquad info_1 = (id_e, info) \\ e = (id_e, \mathsf{webRequest.onBeforeRequest}, \mathsf{none}, info_1, \kappa) \qquad \Sigma' = \Sigma[\psi \Leftarrow \psi']\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{webRequestGet}(\kappa, \cdot, (id_t, id_d, id_n), info, url_{res})) = \Sigma'; e}\ \textsc{WebrequestAPIFire}$$

$$\frac{\begin{array}{l}updates = \mathsf{void}\text{ or }\mathsf{noChange} \\ fresh(id_e{}') \qquad \psi' = \mathsf{ws.beforeSendHeader}(\kappa, id_{cb}, (id_t, id_d, id_n), id_e{}', url, info) \\ e = (id_e{}', \mathsf{webRequest.onBeforeSendHeaders}, \mathsf{none}, info, \kappa)\end{array}}{\mathsf{nextState}(\mathsf{ws.beforeRequest}(\kappa, id_{cb}, (id_t, id_d, id_n), id_e, url, info), updates) = \psi', e}\ \textsc{NS-Ws-BR}(1)$$

$$\frac{\begin{array}{l}updates = \mathsf{Blocked}(\mathsf{redirect}, url_1) \qquad info_2 = \mathsf{updateInfo}(info, updates) \\ fresh(id_e{}') \qquad \psi' = \mathsf{ws.beforeRequest.redirect}(\kappa, id_{cb}, (id_t, id_d, id_n), id_e{}', url_1, info_2) \\ e = (id_e{}', \mathsf{webRequest.onBeforeRedirect}, \mathsf{none}, info_2, \kappa)\end{array}}{\mathsf{nextState}(\mathsf{ws.beforeRequest}(\kappa, id_{cb}, (id_t, id_d, id_n), id_e, url, info), updates) = \psi', e}\ \textsc{NS-Ws-BR}(2)$$

$$\frac{\begin{array}{l}updates = \mathsf{Blocked}(\mathsf{cancel}) \qquad info_2 = \mathsf{updateInfo}(info, updates) \\ id_{cb} = \cdot \qquad fresh(id_e{}') \qquad \psi' = \mathsf{ws.failed}(\kappa, id_{cb}, (id_t, id_d, id_n), id_e{}', url, info_2) \\ e = (id_e{}', \mathsf{webRequest.onErrorOccurred}, \mathsf{none}, info_2, \kappa)\end{array}}{\mathsf{nextState}(\mathsf{ws.beforeRequest}(\kappa, id_{cb}, (id_t, id_d, id_n), id_e, url, info), updates) = \psi', e}\ \textsc{NS-Ws-BR}(3\textsc{a})$$

$$\frac{\begin{array}{l}updates = \mathsf{Blocked}(\mathsf{cancel}) \qquad id_{cb} \neq \cdot \qquad info_2 = \mathsf{updateInfo}(info, updates) \\ fresh(id_e{}') \qquad fresh(id_e{}'') \qquad \psi' = \mathsf{ws.failed}(\kappa, id_{cb}, (id_t, id_d, id_n), id_e{}', url, info_2) \\ e' = (id_e{}', \mathsf{webRequest.onErrorOccurred}, \mathsf{none}, info_2, \kappa) \qquad e'' = (id_e{}'', id_{cb}, \mathsf{none}, info_2, \kappa)\end{array}}{\mathsf{nextState}(\mathsf{ws.beforeRequest}(\kappa, id_{cb}, (id_t, id_d, id_n), id_e, url, info), updates) = \psi', e' :: e''}\ \textsc{NS-Ws-BR}(3\textsc{b})$$

$$\frac{\begin{array}{l}fresh(id_e{}') \qquad \psi' = \mathsf{ws.beforeRequest}(\kappa, id_{cb}, (id_t, id_d, id_n), id_e{}', url, info) \\ e = (id_e{}', \mathsf{webRequest.onBeforeRequest}, \mathsf{none}, info, \kappa)\end{array}}{\mathsf{nextState}(\mathsf{ws.beforeRequest.redirect}(\kappa, id_{cb}, (id_t, id_d, id_n), id_e, url, info), \mathsf{void}) = \psi', e}\ \textsc{NS-Ws-BR}(4)$$

Before the header is sent The state that relates to the blocking event webRequest.onBeforeSendHeaders is ws.beforeSendHeader. Depending on whether there are blocking event handlers, and the result returned by these handlers, the nextState function advances to different states. Here, we also generate an external event denoting that data is send across the network.

$$\frac{\begin{array}{l} \mathit{updates} = \mathsf{void} \vee \mathsf{noChange} \\ \mathit{fresh}(\mathit{id_e}') \qquad \psi' = \mathsf{ws.headerSent}(\kappa, \mathit{id}_{cb}, (\mathit{id}_t, \mathit{id}_d, \mathit{id}_n), \mathit{id_e}', \mathit{url}, \mathit{info}) \\ e = (\mathit{id_e}', \mathsf{webRequest.onSendHeaders}, \mathsf{none}, \mathit{info}, \kappa) \qquad e' = (\_, \mathsf{sendToSever}, \_, (\mathit{url}, \mathit{id_e}'), \kappa') \end{array}}{\mathsf{nextState}(\mathsf{ws.beforeSendHeader}(\kappa, \mathit{id}_{cb}, (\mathit{id}_t, \mathit{id}_d, \mathit{id}_n), \mathit{id}_e, \mathit{url}, \mathit{info}), \mathit{updates}) = \psi', e :: e'} \text{ NS-WS-BSH(1)}$$

The following rule applies when the event handler blocks the header. The blocking handler can rewrite the header.

$$\frac{\begin{array}{l} \mathit{updates} = \mathsf{Blocked}(\mathit{newHeader}, \mathit{header}) \qquad \mathit{info}' = \mathsf{updateInfo}(\mathit{info}, \mathit{updates}) \\ \mathit{fresh}(\mathit{id_e}') \qquad \psi' = \mathsf{ws.headerSent}(\kappa, \mathit{id}_{cb}, (\mathit{id}_t, \mathit{id}_d, \mathit{id}_n), \mathit{id_e}', \mathit{url}, \mathit{info}') \\ e = (\mathit{id_e}', \mathsf{webRequest.onSendHeaders}, \mathsf{none}, \mathit{info}', \kappa) \qquad e' = (\_, \mathsf{sendToSever}, \_, \_, \kappa') \end{array}}{\mathsf{nextState}(\mathsf{ws.beforeSendHeader}(\kappa, \mathit{id}_{cb}, (\mathit{id}_t, \mathit{id}_d, \mathit{id}_n), \mathit{id}_e, \mathit{url}, \mathit{info}), \mathit{updates}) = \psi', e :: e'} \text{ NS-WS-BSH(2)}$$

The following rule applies when one of the event handlers cancels the web request.

$$\frac{\begin{array}{l} \mathit{updates} = \mathsf{Blocked}(\mathsf{Cancel})) \qquad \mathit{info}' = \mathsf{updateInfo}(\mathit{info}, \mathit{updates}) \\ \mathit{id}_{cb} = \cdot \qquad \mathit{fresh}(\mathit{id_e}') \qquad \psi' = \mathsf{ws.failed}(\kappa, \mathit{id}_{cb}, (\mathit{id}_t, \mathit{id}_d, \mathit{id}_n), \mathit{id_e}', \mathit{url}, \mathit{info}') \\ e = (\mathit{id_e}', \mathsf{webRequest.onErrorOccurred}, \mathsf{none}, \mathit{info}', \kappa) \end{array}}{\mathsf{nextState}(\mathsf{ws.beforeSendHeader}(\kappa, \mathit{id}_{cb}, (\mathit{id}_t, \mathit{id}_d, \mathit{id}_n), \mathit{id}_e, \mathit{url}, \mathit{info}), \mathit{updates}) = \psi', e} \text{ NS-WS-BSH(3A)}$$

$$\frac{\begin{array}{l} \mathit{updates} = \mathsf{Blocked}(\mathsf{Cancel})) \qquad \mathit{id}_{cb} \neq \cdot \qquad \mathit{info}' = \mathsf{updateInfo}(\mathit{info}, \mathit{updates}) \\ \mathit{fresh}(\mathit{id_e}') \qquad \mathit{fresh}(\mathit{id_e}'') \qquad \psi' = \mathsf{ws.failed}(\kappa, \mathit{id}_{cb}, (\mathit{id}_t, \mathit{id}_d, \mathit{id}_n), \mathit{id_e}', \mathit{url}, \mathit{info}') \\ e' = (\mathit{id_e}', \mathsf{webRequest.onErrorOccurred}, \mathsf{none}, \mathit{info}', \kappa) \qquad e'' = (\mathit{id_e}'', \mathit{id}_{cb}, \mathsf{none}, \mathit{info}', \kappa) \end{array}}{\mathsf{nextState}(\mathsf{ws.beforeSendHeader}(\kappa, \mathit{id}_{cb}, (\mathit{id}_t, \mathit{id}_d, \mathit{id}_n), \mathit{id}_e, \mathit{url}, \mathit{info}), \mathit{updates}) = \psi', e' :: e''} \text{ NS-WS-BSH(3B)}$$

Receiving header After header is sent, the browser will receive a header from the server. The following two rules are the first place the header is received.

$$\frac{\begin{array}{l} \mathit{fresh}(\mathit{id_e}') \qquad \psi' = \mathsf{ws.headerReceived}(\kappa, \mathit{id}_{cb}, (\mathit{id}_t, \mathit{id}_d, \mathit{id}_n), \mathit{id_e}', \mathit{url}, \mathit{info}') \\ e = (\mathit{id_e}', \mathsf{webRequest.onHeadersReceived}, \mathsf{none}, \mathit{info}', \kappa) \end{array}}{\mathsf{nextState}(\mathsf{headerRecved}(\mathit{url}, \mathit{id}_t, \mathit{id}_d, \mathit{id}_n, \kappa), \cdot) = \psi', e} \text{ NS-WS-HS}$$

Processing header Once the header is received, the browser transitions to a blocking state. *updates* comes from the blocking handlers.

$$\frac{\psi' = \mathsf{ws.headerRecvUpdated}(\kappa, \mathit{id}_{cb}, (\mathit{id}_t, \mathit{id}_d, \mathit{id}_n), \mathit{id}_e, \mathit{url}, \mathit{info}, \mathit{updates})}{\mathsf{nextState}(\mathsf{ws.headerReceived}(\kappa, \mathit{id}_{cb}, (\mathit{id}_t, \mathit{id}_d, \mathit{id}_n), \mathit{id}_e, \mathit{url}, \mathit{info}), \mathit{updates}) = \psi', \cdot} \text{ NS-WS-HR}$$

We allow a blocking event handler to update the header. In the following rule, the web request is for loading an external object (not a page). The header does not contain any CSP. However, the header may contain commands for setting cookies. info and $\mathit{updates}$ contain cookie setting commands from the original header and the blocking event handlers respectively.

$$\frac{\begin{array}{l} \Sigma = (\Psi, \cdots) \\ \Psi = \Psi'' :: \mathsf{ws.headerRecvUpdated}(\kappa, id_{cb}, (id_t, id_d, id_n), id_e, url, \mathit{info}, \mathit{updates}) \\ \mathit{updates} = \mathsf{void} \vee \mathsf{noChange} \vee \mathsf{Blocked}(\mathit{newHdr}, \mathit{header}) \\ \mathit{info}' = \mathsf{updateInfo}(\mathit{info}, \mathit{updates}) \\ \Psi_1 = \mathsf{processCookieCmds}(\kappa, \mathit{info}, \mathit{updates}) \qquad \mathit{header} \text{ is for an object} \\ \mathit{fresh}(id_e{}') \qquad \psi' = \mathsf{ws.responseStarted}(\kappa, id_{cb}, (id_t, id_d, id_n), id_e{}', url, \mathit{info}') \\ e = (id_e{}', \mathsf{webRequest.onResponceStarted}, \mathsf{none}, \mathit{info}', \kappa) \\ \Psi' = \Psi'' :: \psi' :: \Psi_1 \qquad \Sigma' = \Sigma[\Psi \Leftarrow \Psi'] \end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{ws.headerRecvUpdated}(\kappa, id_{cb}, (id_t, id_d, id_n), id_e, url, \mathit{info}, \mathit{updates})) = \Sigma', e} \text{ NS-WS-HR-UPD(1)}$$

If the header is for a document, we extract the CSP from the header, and recompute the label for the document. Note that the CSP is an additional layer of policy applied on top of the policies represented by the original document label ℓ_d. The CSP can only tighten the existing constrains of ℓ_d. We leave this function definition abstract. The generalized CSP could be augmented to include additional information as to how to restrict the label of the doc.

$$\frac{\begin{array}{l} \Sigma = (\Psi, \mathit{Tabs}, \cdots) \\ \Psi = \Psi'' :: \mathsf{ws.headerRecvUpdated}(\kappa, id_{cb}, (id_t, id_d, id_n), id_e, url, \mathit{info}, \mathit{updates}) \\ \mathit{Tabs} = \mathit{Tabs}'' :: t \qquad t = (id_t, \mathit{Docs} :: \mathit{Doc}, \_, \_, \_, \_) \\ \mathit{Doc} = (id_d, \_, \_, \_, \chi, \ell_d) \qquad \mathit{updates} = \mathsf{void} \vee \mathsf{noChange} \vee \mathsf{Blocked}(\mathit{newHdr}, \mathit{header}) \\ \mathit{header} \text{ is for a doc} \qquad \mathit{info}' = \mathsf{updateInfo}(\mathit{info}, \mathit{updates}) \\ \mathit{fresh}(id_e{}') \qquad \psi' = \mathsf{ws.responseStarted}(\ell_d'^{-}, id_{cb}, (id_t, id_d, id_n), id_e{}', url, \mathit{info}') \\ \chi' = \mathsf{getCSP}(\mathit{info}, \mathit{updates}) \\ \Psi_1 = \mathsf{processCookieCmds}(\kappa, \mathit{info}, \mathit{updates}) \qquad \ell_d' = \mathit{computeNewLabel}(\chi', \ell_d) \\ \mathit{Doc}' = \mathit{Doc}[\chi \Leftarrow \chi'][\ell_d \Leftarrow \ell_d'] \qquad t' = t[\mathit{Doc} \Leftarrow \mathit{Doc}'] \qquad \Psi' = \Psi'' :: \psi' :: \Psi_1 \\ \Sigma' = \Sigma[t \Leftarrow t'][\Psi \Leftarrow \Psi'] \qquad e = (id_e{}', \mathsf{webRequest.onResponceStarted}, \mathsf{none}, \mathit{info}', \kappa) \end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{ws.headerRecvUpdated}(\kappa, id_{cb}, (id_t, id_d, id_n), id_e, url, \mathit{info}, \mathit{updates})) = \psi', e} \text{ NS-WS-HR-UPD(2)}$$

The browser enters a failed state if the CSP blocks everything including the content from the to-be-fetched doc itself. This situation is not likely to happen if by default any CSP will support its host page's URL.

$$\frac{\begin{array}{l} \Sigma = (\Psi, \mathit{Tabs}, \cdots) \\ \Psi = \Psi'' :: \mathsf{ws.headerReceived}(\kappa, id_{cb}, (id_t, id_d, id_n), id_e, url, \mathit{info}).\mathit{updates} \\ \mathit{Tabs} = \mathit{Tabs}'' :: t \qquad t = (id_t, \mathit{Docs} :: \mathit{Doc}, \_, \_, \_, \_) \qquad \mathit{Doc} = (id_d, \_, \_, \_, \chi, \ell_d) \\ \mathit{updates} = \mathsf{void} \vee \mathsf{noChange} \vee \mathsf{Blocked}(\mathit{newHdr}, \mathit{header}) \qquad \mathit{header} \text{ is for a doc} \\ \quad \Psi_1 = \mathsf{processCookieCmds}(\kappa, \mathit{info}, \mathit{updates}) \qquad \mathit{info}' = \mathsf{updateInfo}(\mathit{info}, \mathit{updates}) \\ \chi' = \mathsf{getCSP}(\mathit{info}, \mathit{updates}) \qquad \emptyset = \mathit{computeNewLabel}(\chi', \ell_d) \qquad \mathit{fresh}(id_e{}') \\ \psi' = \mathsf{ws.failed}(\kappa, id_{cb}, (id_t, id_d, id_n), id_e{}', url, \mathit{info}_2) \qquad \Psi' = \Psi'' :: \psi' :: \Psi_1 \\ \Sigma' = \Sigma[\Psi \Leftarrow \Psi'] \qquad e = (id_e{}', \mathsf{webRequest.onErrorOccurred}, \mathsf{none}, \mathit{info}_2, \kappa) \end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{ws.headerRecvUpdated}(\kappa, id_{cb}, (id_t, id_d, id_n), id_e, url, \mathit{info}, \mathit{updates})) = \psi', e} \text{ NS-WS-HR-UPD(3)}$$

$$\frac{\begin{array}{l}\Sigma = (\Psi, \mathit{Tabs}, \cdots) \\ \Psi = \Psi'' :: \mathsf{ws.headerReceived}(\kappa, \mathit{id}_{cb}, (\mathit{id}_t, \mathit{id}_d, \mathit{id}_n), \mathit{id}_e, \mathit{url}, \mathit{info}).\mathit{updates} \\ \mathit{Tabs} = \mathit{Tabs}'' :: t \qquad t = (\mathit{id}_t, \mathit{Docs} :: \mathit{Doc}, \_, \_, \_, \_) \qquad \mathit{Doc} = (\mathit{id}_d, \_, \_, \_, \chi, \ell_d) \\ \mathit{updates} = \mathsf{void} \vee \mathsf{noChange} \vee \mathsf{Blocked}(\mathit{newHdr}, \mathit{header}) \qquad \mathit{header} \text{ is for a doc} \\ \Psi_1 = \mathsf{processCookieCmds}(\kappa, \mathit{info}, \mathit{updates}) \qquad \mathit{info}' = \mathsf{updateInfo}(\mathit{info}, \mathit{updates}) \\ \chi' = \mathsf{getCSP}(\mathit{info}, \mathit{updates}) \qquad \emptyset = \mathit{computeNewLabel}(\chi', \ell_d) \\ \mathit{fresh}(\mathit{id}_e') \qquad \mathit{fresh}(\mathit{id}_e'') \qquad \psi' = \mathsf{ws.failed}(\kappa, \mathit{id}_{cb}, (\mathit{id}_t, \mathit{id}_d, \mathit{id}_n), \mathit{id}_e', \mathit{url}, \mathit{info}_2) \\ \Psi' = \Psi'' :: \psi' :: \Psi_1 \qquad \Sigma' = \Sigma[\Psi \Leftarrow \Psi'] \\ e = (\mathit{id}_e', \mathsf{webRequest.onErrorOccurred}, \mathsf{none}, \mathit{info}_2, \kappa) \qquad e' = (\mathit{id}_e'', \mathit{id}_{cb}, \mathsf{none}, \mathit{info}_2, \kappa)\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{ws.headerRecvUpdated}(\kappa, \mathit{id}_{cb}, (\mathit{id}_t, \mathit{id}_d, \mathit{id}_n), \mathit{id}_e, \mathit{url}, \mathit{info}, \mathit{updates})) = \psi', e :: e'} \text{ NS-WS-HR-UPD(4)}$$

Content arriving When the browser receives the content from the server, the web request is completed. The browser goes to the next state. It is a afterFetchedDoc state if the web request loads a page. Else, it is a afterFetched state.

$$\frac{\begin{array}{l}\Sigma = (\Psi, \cdots) \\ \Psi = \Psi_1 :: \mathsf{contentRecved}(\mathit{url}, \mathit{content}, \kappa) :: \mathsf{ws.responseStarted}(\kappa_1, \mathit{id}_{cb}, (\mathit{id}_t, \mathit{id}_d, \mathit{id}_n), \mathit{id}_e, \mathit{url}, \mathit{info}) \\ \mathit{fresh}(\mathit{id}_e') \\ \psi' = \mathsf{ws.completed}(\kappa \uplus_M \kappa_1, \mathit{id}_{cb}, (\mathit{id}_t, \mathit{id}_d, \mathit{id}_n), \mathit{id}_e', \mathit{url}, \mathit{info}', \mathit{content}) \qquad \mathit{id}_{cb} = \cdot \\ e = (\mathit{id}_e', \mathsf{webRequest.onCompleted}, \mathsf{none}, \mathit{info}', \kappa \uplus_M \kappa_1) \qquad \Sigma' = \Sigma[\Psi \Leftarrow \Psi_1 :: \psi']\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{ws.responseStarted}(\kappa_1, \mathit{id}_{cb}, (\mathit{id}_t, \mathit{id}_d, \mathit{id}_n), \mathit{id}_e, \mathit{url}, \mathit{info}), \cdot) = \Sigma', e} \text{ NS-WS-RS(1)}$$

$$\frac{\begin{array}{l}\Sigma = (\Psi, \cdots) \\ \Psi = \Psi_1 :: \mathsf{contentRecved}(\mathit{url}, \mathit{content}, \kappa) :: \mathsf{ws.responseStarted}(\kappa_1, \mathit{id}_{cb}, (\mathit{id}_t, \mathit{id}_d, \mathit{id}_n), \mathit{id}_e, \mathit{url}, \mathit{info}) \\ \mathit{fresh}(\mathit{id}_e') \\ \mathit{fresh}(\mathit{id}_e'') \qquad \psi' = \mathsf{ws.completed}(\kappa \uplus_M \kappa_1, \mathit{id}_{cb}, (\mathit{id}_t, \mathit{id}_d, \mathit{id}_n), \mathit{id}_e', \mathit{url}, \mathit{info}', \mathit{content}) \\ \mathit{id}_{cb} \neq \cdot \qquad e = (\mathit{id}_e', \mathsf{webRequest.onCompleted}, \mathsf{none}, \mathit{info}', \kappa \uplus_M \kappa_1) \\ e' = (\mathit{id}_e'', \mathit{id}_{cb}, \mathsf{none}, \mathit{info}', \kappa \uplus_M \kappa_1) \qquad \Sigma' = \Sigma[\Psi \Leftarrow \Psi_1 :: \psi']\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{ws.responseStarted}(\kappa_1, \mathit{id}_{cb}, (\mathit{id}_t, \mathit{id}_d, \mathit{id}_n), \mathit{id}_e, \mathit{url}, \mathit{info}), \cdot) = \Sigma', e :: e'} \text{ NS-WS-RS(2)}$$

$$\frac{\begin{array}{l}\text{“}\mathit{doc}\text{”} \in \mathit{info} \qquad \mathsf{framePolicy} \in \mathit{info} \qquad \psi' = \mathsf{afterFetchedDoc}(\kappa, \mathit{id}_t, \mathit{id}_d, \mathsf{framePolicy}, \mathit{content}) \\ \mathit{fresh}(\mathit{id}_e') \qquad e = (\mathit{id}_e', \mathsf{DOM.beforeLoad}, \mathsf{none}, \mathit{info}, \kappa)\end{array}}{\mathsf{nextState}(\mathsf{ws.completed}(\kappa, \mathit{id}_{cb}, (\mathit{id}_t, \mathit{id}_d, \mathit{id}_n), \mathit{id}_e, \mathit{url}, \mathit{info}, \mathit{content}), \cdot) = \psi', e} \text{ NS-WS-C}$$

$$\frac{\begin{array}{l}\mathit{id}_n \neq \cdot \wedge \text{“}\mathit{doc}\text{”} \notin \mathit{info} \qquad \psi' = \mathsf{afterFetched}(\kappa, \mathit{id}_t, \mathit{id}_d, \mathit{id}_n, \mathit{content}) \\ \mathit{fresh}(\mathit{id}_e') \qquad e = (\mathit{id}_e', \mathsf{DOM.beforeLoad}, \mathsf{none}, \mathit{info}, \kappa)\end{array}}{\mathsf{nextState}(\mathsf{ws.completed}(\kappa, \mathit{id}_{cb}, (\mathit{id}_t, \mathit{id}_d, \mathit{id}_n), \mathit{id}_e, \mathit{url}, \mathit{info}, \mathit{content}), \cdot) = \psi', e} \text{ NS-WS-C-NODE}$$

3.7.3 Content Loading

Content fetched from server is loaded to a node. The node is not a script node.

$$\frac{\begin{array}{l}\Sigma = (\Psi :: \psi, \mathit{Tabs}, \mathit{Exts}, \mathit{ExtCoreRs}, \cdots) \\ \mathit{Tabs} = t :: \mathit{Tabs}_1 \qquad t = (id_t, \mathit{Docs} :: \mathit{Doc}, \mathsf{url}_1, \_, \_, \ell_{tab}) \\ \psi = \mathsf{afterFetched}(\kappa, id_t, id_d, id_n, \mathit{content}) \qquad \mathit{Doc} = (id_d, \mathit{nodes}, \cdots) \\ \mathit{node} \in \mathit{nodes} \qquad \mathit{node} = (id_n, \mathit{attributes}, \cdot, \mathit{content}_n, \ell_n) \qquad \mathit{type} \in \mathit{attributes} \\ \mathit{type} \neq \mathsf{script} \qquad \ell'_n = \kappa \rhd_{tnt} \ell_n \qquad \mathit{node}' = \mathit{node}[\mathit{content}_n \Leftarrow \mathit{content}, \ell_n \Leftarrow \ell'_n] \\ \mathit{nodes}' = \mathit{nodes}[\mathit{node} \Leftarrow \mathit{node}'] \qquad \mathit{Doc}' = \mathit{Doc}[\mathit{nodes} \Leftarrow \mathit{nodes}'] \\ \psi' = \cdot \qquad t' = t[\mathit{Doc} \Leftarrow \mathit{Doc}'] \qquad \Sigma' = \Sigma[t \Leftarrow t'][\psi \Leftarrow \psi'] \end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{afterFetched}(\kappa, id_t, id_d, id_n, \mathit{content}) = \Sigma'; \cdot} \ \textsc{ContentLoading}$$

A script is loaded to a node. The browser extracts event handlers from the script.

$$\frac{\begin{array}{l}\Sigma = (\Psi :: \psi, \mathit{Tabs}, \mathit{Exts}, \mathit{ExtCoreRs}, \cdots) \\ \mathit{Tabs} = t :: \mathit{Tabs}_1 \qquad t = (id_t, \_, \mathit{Docs} :: \mathit{Doc}, \_, \mathit{EventHandlers}, \cdots) \\ \psi = \mathsf{afterFetched}(\kappa, id_t, id_d, id_n, \mathit{content}) \qquad \mathit{Doc} = (id_d, \mathit{nodes}, \cdots) \\ \mathit{node} \in \mathit{nodes} \qquad \mathit{node} = (id_n, \mathit{attributes}, \cdot, \mathit{content}_n, \ell_n) \qquad \mathit{type} \in \mathit{attributes} \\ \mathit{type} = \mathsf{script} \qquad \ell'_n = \kappa \rhd_{tnt} \ell_n \qquad \mathit{node}' = \mathit{node}[\mathit{content}_n \Leftarrow \mathit{content}, \ell_n \Leftarrow \ell'_n] \\ \mathit{nodes}' = \mathit{nodes}[\mathit{node} \Leftarrow \mathit{node}'] \qquad \mathit{Doc}' = \mathit{Doc}[\mathit{nodes} \Leftarrow \mathit{nodes}'] \\ \mathit{EventHandlers}' = \mathit{registerHandlers}(\mathit{EventHandlers}, \mathit{content}) \\ \psi' = \cdot \qquad t' = t[\mathit{Doc} \Leftarrow \mathit{Doc}'][\mathit{EventHandlers} \Leftarrow \mathit{EventHandlers}'] \\ \Sigma' = \Sigma[t \Leftarrow t'][\psi \Leftarrow \psi'] \\ \mathit{fresh}(id_e) \qquad e = (id_e, \mathsf{DOM.scriptLoaded}, (id_t, id_d, id_n), \ell_n'^{-}) \end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{afterFetched}(\kappa, id_t, id_d, id_n, \mathit{content}) = \Sigma'; e} \ \textsc{ContentLoadingScript}$$

After fetching the content for a page, the browser is ready to create the page's DOM tree.

$$\frac{\begin{array}{l}\mathit{fresh}(id) \qquad \psi' = \mathsf{readytoCreateDoc}(id_t, id_d, id, \mathit{content}, \kappa) \\ e = (id, \mathsf{webNavigation.onCommitted}, \mathsf{none}, \mathit{info}, \kappa) \end{array}}{\mathsf{nextState}(\Sigma, \mathsf{afterFetchedDoc}(\kappa, id_t, id_d, id_n, id_e, \mathit{content})) = \psi'; e} \ \textsc{ContentLoadingDoc}$$

Create the DOM tree Next, the browser creates the DOM tree and injects content scripts. DOM tree creation contains three phases, DocBegin, DocEnd, and DocIdle. At each phase, the browser only injects the content scripts that are set to run in this phase.

$$\frac{\begin{array}{l}\Sigma = (\Psi, \mathit{Tabs} :: t, \mathit{Exts}, \mathit{ExtCoreRs}, \cdots) \\ \Psi = \Psi_1 :: \psi \qquad t = (id_t, \mathit{Docs} :: \mathit{Doc}, \_, \_, \_, \_) \qquad \mathit{Doc} = (id_d, \mathit{url}, \mathit{nodes}, \mathit{DocCSs}, \chi, \ell) \\ \psi = \mathsf{readytoCreateDoc}(\kappa, id_t, id_d, \mathit{content}) \qquad \psi_1 = \mathsf{DocBegin}(\kappa, id_t, id_d, \mathit{content}) \\ \mathit{DocCSs}_1 = \mathit{injectCSAll}(\mathit{Exts}, \mathit{progInjCSs}, id_t, id_d, \mathsf{DocBegin}) \\ \mathit{Doc}' = \mathit{Doc}[\mathit{DocCSs} \Leftarrow \mathit{DocCS}_1] \qquad t' = t[\mathit{Doc} \Leftarrow \mathit{Doc}'] \qquad \Sigma' = \Sigma[t \Leftarrow t'][\Psi \Leftarrow \Psi_1 :: \psi_1] \end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{readytoCreateDoc}(\kappa, id_t, id_d, \mathit{content})) = \Sigma'; \cdot} \ \textsc{CreateDoc}(1)$$

In CREATEDOC(2), a DOM tree is constructed as the browser renders a page. A node could load a single piece of external content (e.g., an image or a text file) or an iframed page. When a node that requires a single piece of external content is created, a corresponding readyToFetch state is generated as well. The node's label and the state's label are computed based on the node's parent's label. When an iframe node is created, the system will generate a corresponding readyToFetchDoc state. The node's label is computed based on its parent's label and the iframe policy set by the parent node. The readyToFetchDoc state's label is computed merely based on the node's label. After generating the DOM nodes, and the readyToFetch and readyToFetchDoc states , the document's status proceeds from DocBegin to DocEnd without loading external objects and subpages.

$$\frac{\begin{array}{l}\Sigma = (\Psi, \mathit{Tabs} :: t, \mathit{Exts}, \mathit{ExtCoreRs}, \cdots) \quad \Psi = \Psi_1 :: \psi \\ \psi = \mathsf{DocBegin}(\kappa, id_t, id_d, \mathit{content}) \quad (\Psi_2, t') = \mathit{domTreeCreation}(\mathit{content}, t) \\ t' = (id_t, \mathit{Docs} :: \mathit{Doc}, \_, \_, \_, \_) \quad \mathit{Doc} = (id_d, \mathit{url}, \mathit{nodes}, \mathit{DocCSs}, \chi, \ell) \\ \psi_1 = \mathsf{DocEnd}(\kappa, id_t, id_d) \quad \mathit{DocCSs}_1 = \mathit{injectCSAll}(\mathit{Exts}, \mathit{progInjCSs}, id_t, id_d, \mathsf{DocEnd}) \\ \mathit{Doc}' = \mathit{Doc}[\mathit{DocCSs} \Leftarrow \mathit{DocCSs} :: \mathit{DocCS}_1] \\ t_2 = t'[\mathit{Doc} \Leftarrow \mathit{Doc}'] \quad \Sigma' = \Sigma[t \Leftarrow t_2][\Psi \Leftarrow \Psi_1 :: \psi_1 :: \Psi_2]\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{DocBegin}(\kappa, id_t, id_d, \mathit{content})) = \Sigma'; \cdot} \text{ CREATEDOC(2)}$$

$$\frac{\begin{array}{l}\Sigma = (\Psi, \mathit{Tabs} :: t, \mathit{Exts}, \mathit{ExtCoreRs}, \cdots) \quad \Psi = \Psi_1 :: \psi \quad t = (id_t, \mathit{Docs} :: \mathit{Doc}, \_, \_, \_, \_) \\ \mathit{Doc} = (id_d, \mathit{url}, \mathit{nodes}, \mathit{DocCSs}, \chi, \ell) \quad \psi = \mathsf{DocEnd}(\kappa, id_t, id_d) \\ \psi_1 = \mathsf{DocIdle}(\kappa, id_t, id_d) \quad \mathit{DocCSs}_1 = \mathit{injectCSAll}(\mathit{Exts}, \mathit{progInjCSs}, id_t, id_d, \mathsf{DocIdle}) \\ \mathit{Doc}' = \mathit{Doc}[\mathit{DocCSs} \Leftarrow \mathit{DocCS} :: \mathit{DocCS}_1] \\ t' = t[\mathit{Doc} \Leftarrow \mathit{Doc}'] \quad \Sigma' = \Sigma[t \Leftarrow t'][\Psi \Leftarrow \Psi_1 :: \psi_1]\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{DocEnd}(\kappa, id_t, id_d)) = \Sigma'; \cdot} \text{ CREATEDOC(3)}$$

Content Script Injection When document state changes, both the scripts in the extensions and programmatically injected scripts get a chance to be injected to the page. $\mathit{injectCSExts}(\mathit{Exts}, id_t, id_d, \mathit{docState})$ extracts the content scripts that can be executed in the page with tab ID id_t and Doc ID id_d from extensions Exts. If an extension is active, the content scripts to be injected are extracted from the extension's static content scripts. If an extension is inactive, none of its content scripts will be injected to the page.

$$\frac{\begin{array}{c}\mathit{DocCSs}_1 = \mathit{injectCSExts}(\mathit{Exts}, id_t, id_d, \mathit{docState}) \\ \mathit{DocCSs}_2 = \mathit{injectCSs}(\mathit{progInjCSs}, id_t, id_d, \mathit{docState})\end{array}}{\mathit{injectCSAll}(\mathit{Exts}, \mathit{progInjCSs}, id_t, id_d, \mathit{docState}) = \mathit{DocCSs}_1 :: \mathit{DocCSs}_2} \text{ INJECTCSSALL}$$

$$\frac{}{\mathit{injectCSExts}(\cdot, id_t, id_d, \mathit{docState}) = \cdot} \text{ INJECTCSEXTS-NIL}$$

$$\frac{\begin{array}{c}\mathit{DocCSs}_1 = \mathit{injectCSExts}(\mathit{Exts}, id_t, id_d, \mathit{docState}) \\ \mathit{DocCSs}_2 = \mathit{injectCSExt}(\mathit{Ext}, id_t, id_d, \mathit{docState})\end{array}}{\mathit{injectCSExts}(\mathit{Exts} :: \mathit{Ext}, id_t, id_d, \mathit{docState}) = \mathit{DocCSs}_1 :: \mathit{DocCSs}_2} \text{ INJECTCSEXTS-CONS}$$

$$\frac{\mathit{Ext} = (\_, \_, \mathit{ExtCSs}, \_, \mathsf{active}, \ell) \quad \mathit{DocCSs} = \mathit{injectCSs}(\mathit{ExtCSs}, id_t, id_d, \mathit{docState})}{\mathit{injectCSExt}(\mathit{Ext}, id_t, id_d, \mathit{docState}) = \mathit{DocCSs}} \text{ INJECTCSEXT-ACTIVE}$$

$$\frac{\mathit{Ext} = (\_, \_, \mathit{ExtCSs}, \_, \mathsf{inactive}, \ell)}{\mathit{injectCSExts}(\mathit{Ext}, id_t, id_d, \mathit{docState}) = \cdot} \text{ INJECTCSEXT-INACTIVE}$$

$$\frac{}{\mathit{injectCSs}(\cdot, id_t, id_d, \mathit{docState}) = \cdot}\ \textsc{InjectCSs-Nil}$$

$$\frac{\begin{array}{l}\mathit{DocCS} = \mathit{injectCS}(\mathit{ExtCS}, id_t, \ell_d, \mathit{docState})\\ \mathit{DocCSs} = \mathit{injectCSs}(\mathit{ExtCSs}, id_t, \ell_d, \mathit{docState})\end{array}}{\mathit{injectCSs}(\mathit{ExtCSs} :: \mathit{ExtCS}, id_t, id_d, \mathit{docState}) = \mathit{DocCS} :: \mathit{DocCSs}}\ \textsc{InjectCSs-Cons}$$

The content script stored in an installed extension has an empty tab ID. If it is allowed to be injected, we fill in the hosting tab ID in the place of the empty tab ID. The programmatically injected scripts have a tab ID associated. This tab ID has to be the same as the tab ID argument. Similarly, we compute the label for the script and associate a fresh ID with it. We do not inject a script if the labels prevent such injection.

$$\frac{\begin{array}{l}\mathit{ExtCS} = (id_{ext}, id_{cs}, \cdot, \Gamma, \mathit{cmd}, \mathit{EventHandlers}, \mathit{runat}, \ell_1) \qquad \mathit{runat} = \mathit{docState}\\ \mathit{Doc} = (id_d, \mathit{url}, \_, \_, \chi, \ell_d) \qquad \ell_1' = \mathit{computeResLab}(\mathit{url}, {\ell_d}^-, \chi, \ell_1, id_{ext})\\ \mathit{fresh}(id) \qquad \mathit{DocCS} = (id_{ext}, id_{cs}, id, \Gamma, \mathit{cmd}, \mathit{EventHandlers}, \ell_1')\end{array}}{\mathit{injectCSs}(\mathit{ExtCS}, id_t, id_d, \mathit{docState}) = \mathit{DocCS}}\ \textsc{InjectCSs-Inj-Ext}$$

$$\frac{\begin{array}{l}\mathit{ExtCS} = (id_{ext}, id_{cs}, id_t, \Gamma, \mathit{cmd}, \mathit{EventHandlers}, \mathit{runat}, \ell_1) \qquad \mathit{runat} = \mathit{docState}\\ \mathit{Doc} = (id_d, \mathit{url}, \_, \_, \chi, \ell_d) \qquad \ell_1' = \mathit{computeResLab}(\mathit{url}, {\ell_d}^-, \chi, \ell_1, id_{ext})\\ \mathit{fresh}(id) \qquad \mathit{DocCS} = (id_{ext}, id_{cs}, id, \Gamma, \mathit{cmd}, \mathit{EventHandlers}, \ell_1')\end{array}}{\mathit{injectCSs}(\mathit{ExtCS}, id_t, id_d \mathit{docState}) = \mathit{DocCS}}\ \textsc{InjectCSs-Inj}$$

$$\frac{\begin{array}{l}\mathit{ExtCS} = (id_{ext}, id_{cs}, {id_t}', \Gamma, \mathit{cmd}, \mathit{EventHandlers}, \mathit{runat}, \ell_1)\\ \mathit{Doc} = (id_d, \mathit{url}, \_, \_, \chi, \ell_d) \qquad \ell_1' = \mathit{computeResLab}(\mathit{url}, {\ell_d}^-, \chi, \ell_1, id_{ext})\\ \mathit{runat} \neq \mathit{docState} \text{ or } (id_t \neq {id_t}' \text{ and } {id_t}' \neq \cdot) \text{ or } \ell_1' = \emptyset\end{array}}{\mathit{injectCSs}(\mathit{ExtCS}, id_t, id_d, \mathit{docState}) = \cdot}\ \textsc{InjectCSs-Discard}$$

3.8 Browser-specific Rules

3.8.1 Rules for Accessing DOM

DOM node details Scripts can not only read and write DOM nodes but also change the tree structure of DOM by inserting, deleting, and moving node pointers. We distinguish between labels related to the structure of the DOM tree and the label for the content stored in the node. In the following, a node contains four labels in its attribute field: ℓ_{parent} is the label for parent point, $\ell_{children}$ the label for the children points, ℓ_{pre} the label for the previous sibling pointer, and ℓ_{succ} the label for the succeeding sibling pointer. For a script to change a node's parent pointer, the script's label must be lower or equal to the target node's ℓ_{parent}.

$$\begin{array}{lll} \mathit{Node} & \mathit{node} & ::= \ (id, \mathit{attributes}, \mathit{nodes}, \mathit{content}, \ell)\\ \mathit{Attributes} & \mathit{attributes} & ::= \ (\mathit{type}, \mathit{url}, \ell_{parent}, \ell_{children}, \ell_{pre}, \ell_{succ}, \cdots) \end{array}$$

DomGetNodePointer Scripts can access DOM nodes in the same document. Scripts can refer to one or a list of elements by attributes (e.g., node ID and tag) through the document.getElements$*$ API. For example, with "var myList = document.getElementsByTagName("p")", myList refers to the pointers of the nodes with tag "p". To get the value of a specific node property *prop* of a node, one can use a command like "value = myList[0].prop", where $\mathit{myList}[0]$ is the pointer to the target node.

In this rule, we model the document.getElementsBy* APIs with domAPI.getNodePointer$(id_t, id_d, \mathit{query})$. id_t and id_d indicate to which tab and doc the caller script belongs. Different document.getElementsBy* APIs match to different $query$s. E.g., for “var myList = document.getElementsByTagName(”p”);”, $query$ should be “TagName, p”. $\mathsf{pointersOf}(nodes)$ denotes the pointers of $nodes$.

A script could attemp to leak secrets by affecting the return results of such queries. For instance, if a secret script deleted a node with tag “p”, an attacker that used the API to return all pointers to nodes with tag “p” can observe the elements returned and know whether the first script have deleted the element. Mitigate this leak, we assume this is a lable assigned to each query. When the return results change, the label of the qeury will reflect that change. For instance, there is a global label for elements with tag “p”. When one program with label l_1 deletes an element with tag “p”, then the label for the quey is updated to be tainted by the labell_1. In practice, this data structure needs to be built and maintained by the browser runrume.

If the caller’s label is lower than the label of the query, the system returns the matching nodes’ pointers. The caller’s label gets tainted by the matching nodes’ labels. Otherwise, the execution gets stuck.

$$\frac{\begin{array}{l}\Sigma_{ctx}(id) = (\Psi, Tabs :: t_{ctx}(id), \cdots) \\ \ell = \mathsf{ctxOfId}(\Sigma_{ctx} \,\|\, \Gamma, \mathsf{let}\ x = \mathsf{domAPI.getNodePointer}(id_t, id_d, query)\ \mathsf{in}\ cmd \,\|_{id}, id) \\ \kappa = \ell^* \qquad t = (id_t, Docs :: Doc, \cdots) \\ Doc = (id_d, \cdots) \qquad \kappa_Q = \mathsf{labOfQ}(query, Doc) \qquad {\kappa_Q}^- \sqsubseteq \kappa \\ nodes_1 = QueryNodes(query, Doc) \qquad \Sigma'_{ctx}(id) = \mathsf{update}(\Sigma_{ctx}(id), \kappa_Q \rhd_{tnt} \ell)\end{array}}{\begin{array}{l}\Sigma_{ctx} \,\|\, \Gamma, \mathsf{let}\ x = \mathsf{domAPI.getNodePointer}(id_t, id_d, query)\ \mathsf{in}\ cmd \,\|_{id} \\ \xrightarrow{\mathsf{call}(\mathsf{domAPI.getNodePointer}, (id_t, id_d, query), \mathsf{pointersOf}(nodes_1), \kappa_Q)} \Sigma'_{ctx} \,\|\, \Gamma, cmd[\mathsf{pointersOf}(nodes_1)/x] \,\|_{id}; \cdot\end{array}}\ \textsc{DomGetNodePointer}$$

$$\frac{\begin{array}{l}\Sigma_{ctx}(id) = (\Psi, Tabs :: t_{ctx}(id), \cdots) \\ \ell = \mathsf{ctxOfId}(\Sigma_{ctx} \,\|\, \Gamma, \mathsf{let}\ x = \mathsf{domAPI.getNodePointer}(id_t, id_d, query)\ \mathsf{in}\ cmd \,\|_{id}, id) \\ \kappa = \ell^* \qquad t = (id_t, Docs :: Doc, \cdots) \\ Doc = (id_d, \cdots) \qquad \kappa_Q = \mathsf{labOfQ}(query, Doc) \qquad {\kappa_Q}^- \not\sqsubseteq \kappa\end{array}}{\Sigma_{ctx} \,\|\, \Gamma, \mathsf{let}\ x = \mathsf{domAPI.getNodePointer}(id_t, id_d, query)\ \mathsf{in}\ cmd \,\|_{id} \quad \xrightarrow{\tau} \mathsf{stuck}}\ \textsc{DomGetNodePointerS}$$

DomNodeRead Given a pointer to a node, a property value of the node is read through a command like “value = myList[0].prop”. $name$ is the property name. The system returns the value if the node’s label is lower than κ. The caller’s label gets tainted by the node’s label. Or else, the system gets stuck.

$$\frac{\begin{array}{l}\Sigma_{ctx}(id) = (\Psi, Tabs :: t_{ctx}(id), \cdots) \\ \ell = \mathsf{ctxOfId}(\Sigma_{ctx} \,\|\, \Gamma, \mathsf{let}\ x = \mathsf{domAPI.readNode}(id_t, id_d, id_n, name)\ \mathsf{in}\ cmd \,\|_{id}, id) \\ \kappa = \ell^* \qquad t = (id_t, Docs :: Doc, \cdots) \\ Doc = (id_d, \cdots) \qquad node \in Doc \qquad node = (id_n, \cdots, \ell_n) \qquad {\ell_n}^- \sqsubseteq \kappa \\ result = \mathsf{getPropertyValue}(node, name) \qquad \Sigma'_{ctx}(id) = \mathsf{update}(\Sigma_{ctx}(id), {\ell_n}^- \rhd_{tnt} \ell)\end{array}}{\begin{array}{l}\Sigma_{ctx} \,\|\, \Gamma, \mathsf{let}\ x = \mathsf{domAPI.readNode}(id_t, id_d, id_n, name)\ \mathsf{in}\ cmd \,\|_{id} \\ \xrightarrow{\mathsf{call}(\mathsf{domAPI.readNode}, (id_t, id_d, id_n, name), result, {\ell_n}^-)} \Sigma'_{ctx} \,\|\, \Gamma, cmd[result/x] \,\|_{id}; \cdot\end{array}}\ \textsc{DomNodeRead}$$

DOMNODEWRITE(1) A script is allowed to update the content of a node if its label is lower than the node’s label. In this rule, the script updates the whole node, and can assign the node with a new label which should be lower than the script’s label. DomNodeWrite API does not return any value.

$$\dfrac{\begin{array}{l} \Sigma_{ctx}(id) = (\Psi, \mathit{Tabs} :: t_{\mathit{ctx}}(id), \cdots) \\ \ell = \mathsf{ctxOfId}(\Sigma_{ctx} \,\|\, \Gamma, \mathsf{domAPI.writeNode}(id_t, id_d, id_n, \mathit{content}', \kappa_1) \,\|_{id}, id) \qquad \kappa = \ell^{-} \\ t = (id_t, \_, \mathit{Docs} :: \mathit{Doc}, \cdots) \\ \mathit{Doc} = (id_d, \_, \mathit{nodes}, \_, \_) \qquad \mathit{node} \in_b \mathit{nodes} \qquad \mathit{node} = (id_n, \_, \_, \mathit{content}, \ell_{node}) \\ \kappa \sqsubseteq {\ell_{node}}^{*} \qquad \mathit{node}' = (id_n, \_, \_, \mathit{content}', \kappa_1 \rhd_{tnt} \ell_{node}) \\ \mathit{Doc}' = \mathit{Doc}[\mathit{node} \Leftarrow \mathit{node}'] \qquad t' = t[\mathit{Doc} \Leftarrow \mathit{Doc}'] \qquad \Sigma_{ctx}(id)' = \Sigma_{ctx}(id)[t \Leftarrow t'] \end{array}}{\begin{array}{l} \Sigma_{ctx} \,\|\, \Gamma, \mathsf{domAPI.writeNode}(id_t, id_d, id_n, \mathit{content}', \kappa_1) \,\|_{id} \\ \xrightarrow{\mathsf{call}(\mathsf{domAPI.writeNode}, (id_t, id_d, id_n, \mathit{content}', \kappa_1), \cdot, \kappa_1)} \Sigma'_{ctx} \,\|\, \Gamma, \mathsf{skip} \,\|_{id}; \cdot \end{array}} \textsc{DomNodeWrite}(1)$$

DOMNODEWRITE(2) A script fails to write something to a node by assignment. Label check fails, or the target node does not exist. DomNodeWrite API does not return any value.

$$\dfrac{\begin{array}{l} \Sigma_{ctx}(id) = (\Psi, \mathit{Tabs} :: t_{\mathit{ctx}}(id), \cdots) \\ \ell = \mathsf{ctxOfId}(\Sigma_{ctx} \,\|\, \Gamma, \mathsf{domAPI.writeNode}(id_t, id_d, id_n, \mathit{content}', \kappa_1) \,\|_{id}, id) \qquad \kappa = \ell^{-} \\ t = (id_t, \_, \mathit{Docs} :: \mathit{Doc}, \cdots) \qquad \mathit{Doc} = (id_d, \_, \mathit{nodes}, \_, \_) \\ (\nexists \mathit{node} \in \mathit{nodes} \text{ s.t. } \mathit{node} = (id_n, \cdots)) \\ \quad \text{or } (\exists \mathit{node} \in \mathit{nodes} \text{ s.t. } \mathit{node} = (id_n, \cdots, \ell_{node}) \text{ and } \kappa \not\sqsubseteq {\ell_{node}}^{*}) \end{array}}{\begin{array}{l} \Sigma_{ctx} \,\|\, \Gamma, \mathsf{domAPI.writeNode}(id_t, id_d, id_n, \mathit{content}', \kappa_1) \,\|_{id} \\ \xrightarrow{\mathsf{call}(\mathsf{domAPI.writeNode}, (id_t, id_d, id_n, \mathit{content}', \kappa_1), \cdot, \kappa_1)} \Sigma_{ctx} \,\|\, \Gamma, \mathsf{skip} \,\|_{id}; \cdot \end{array}} \textsc{DomNodeWrite}(2)$$

DomGetChildren This API returns the set of pointers of the children of a node. The caller's label must be lower than or equal to the children point label of the node; otherwise, the program is stuck.

$$\dfrac{\begin{array}{l} \Sigma_{ctx}(id) = (\Psi, \mathit{Tabs} :: t_{\mathit{ctx}}(id), \cdots) \\ \ell = \mathsf{ctxOfId}(\Sigma_{ctx} \,\|\, \Gamma, \mathsf{let}\ x = \mathsf{domAPI.getChildren}(id_t, id_d, id_n) \ \mathsf{in}\ \mathit{cmd} \,\|_{id}, id) \\ \kappa = \ell^{*} \qquad t = (id_t, \mathit{Docs} :: \mathit{Doc}, \cdots) \\ \mathit{Doc} = (id_d, \cdots) \qquad \mathit{node} \in \mathit{Doc} \qquad \mathit{node} = (id_n, \mathit{nodes}, (\cdots, \ell_{children}, \cdots), \cdots, \ell_n) \\ {\ell_{children}}^{-} \sqsubseteq \kappa \qquad \Sigma'_{ctx}(id) = \mathsf{update}(\Sigma_{ctx}(id), {\ell_{children}}^{-} \rhd_{tnt} \ell) \end{array}}{\begin{array}{l} \Sigma_{ctx} \,\|\, \Gamma, \mathsf{let}\ x = \mathsf{domAPI.getChildren}(id_t, id_d, id_n) \ \mathsf{in}\ \mathit{cmd} \,\|_{id} \\ \xrightarrow{\mathsf{call}(\mathsf{domAPI.getChildren}, (id_t, id_d, id_n), \mathsf{pointersOf}(\mathit{nodes}), {\ell_{children}}^{-})} \Sigma'_{ctx} \,\|\, \Gamma, \mathit{cmd}[\mathsf{pointersOf}(\mathit{nodes})/x] \,\|_{id}; \cdot \end{array}} \textsc{DomGetChildren}$$

DomGetParent This API returns a node's parent pointer, if the caller's label is lower than or equal to the node's parent pointer label. Otherewise, the execution gets stuck.

$$\textsc{DomGetParent}\quad \dfrac{\begin{array}{l} \Sigma_{ctx}(id) = (\Psi, \mathit{Tabs} :: t_{\mathit{ctx}}(id), \cdots) \\ \ell = \mathsf{ctxOfId}(\Sigma_{ctx} \,\|\, \Gamma, \mathsf{let}\ x = \mathsf{domAPI.getParent}(id_t, id_d, id_n) \ \mathsf{in}\ \mathit{cmd} \,\|_{id}, id) \qquad \kappa = \ell^{*} \\ t = (id_t, \mathit{Docs} :: \mathit{Doc}, \cdots) \qquad \mathit{Doc} = (id_d, \cdots) \qquad \mathit{node} \in \mathit{Doc} \qquad \mathit{node} = (id_n, (\cdots, \ell_{parent}, \cdots), \cdots, \ell_n) \\ {\ell_{parent}}^{-} \sqsubseteq \kappa \qquad p = \mathsf{parentOf}(id, \mathit{Doc}) \qquad \Sigma'_{ctx}(id) = \mathsf{update}(\Sigma_{ctx}(id), {\ell_{parent}}^{-} \rhd_{tnt} \ell) \end{array}}{\begin{array}{l} \Sigma_{ctx} \,\|\, \Gamma, \mathsf{let}\ x = \mathsf{domAPI.getParent}(id_t, id_d, id_n) \ \mathsf{in}\ \mathit{cmd} \,\|_{id} \\ \xrightarrow{\mathsf{call}(\mathsf{domAPI.getParent}, (id_t, id_d, id_n), p, {\ell_{parent}}^{-})} \Sigma'_{ctx} \,\|\, \Gamma, \mathit{cmd}[\mathsf{pointersOf}(id_{parent})/x] \,\|_{id}; \cdot \end{array}}$$

DomGetPreSibling Get a node's previous sibling.

$$\frac{\begin{array}{l}\textsc{DomGetPreSibling}\\ \Sigma_{ctx}(id) = (\Psi, \mathit{Tabs} :: t_{\mathit{ctx}}(id), \cdots)\\ \ell = \mathsf{ctxOfId}(\Sigma_{ctx} \llbracket \Gamma, \mathsf{let}\ x = \mathsf{domAPI.getPreviousSibling}(id_t, id_d, id_n)\ \mathsf{in}\ cmd \rrbracket_{id}, id)\\ \kappa = \ell^* \qquad t = (id_t, \mathit{Docs} :: \mathit{Doc}, \cdots) \qquad \mathit{Doc} = (id_d, \cdots)\\ node \in \mathit{Doc} \qquad node = (id_n, (\cdots, \ell_{pre}, \cdots), \cdots, \ell_n) \qquad {\ell_{pre}}^- \sqsubseteq \kappa \qquad p = \mathsf{preSiblingOf}(id_n, \mathit{Doc})\\ \Sigma'_{ctx}(id) = \mathsf{update}(\Sigma_{ctx}(id), {\ell_{pre}}^- \rhd_{tnt} \ell)\end{array}}{\begin{array}{l}\Sigma_{ctx} \llbracket \Gamma, \mathsf{let}\ x = \mathsf{domAPI.getPreviousSibling}(id_t, id_d, id_n)\ \mathsf{in}\ cmd \rrbracket_{id}\\ \xrightarrow{\mathsf{call}(\mathsf{domAPI.getPreviousSibling}, (id_t, id_d, id_n), p, {\ell_{pre}}^-)} \Sigma'_{ctx} \llbracket \Gamma, cmd[p/x] \rrbracket_{id}; \cdot\end{array}}$$

DOMAppendChildren Append a child node to a node $node$. $node_1$ is the to-be-appended node. $node_1$ has a label ℓ_1 which can be lower than κ. When the domAPI.appendChild API call is issued, in the rule FIREAPICALL-S2, we make sure that labOf$(node_1)$ is lower than κ.

DOMAPPENDCHILDREN: Successfully appends $node_1$ to $node$. There are three cases: a) the node will not load contents from a server; b) the node will load an external object, for example, an image; c) the node will load a page into an iframe. For case 1b, a readyToFetch state will be generated. For case 1c, a readyToFetchDoc state will be generated.

$$\frac{\begin{array}{l}\textsc{DomAppendChildren(a)}\\ \Sigma_{ctx}(id) = (\Psi, \mathit{Tabs} :: t_{\mathit{ctx}}(id), \cdots)\\ \ell_1 = \mathsf{ctxOfId}(\Sigma_{ctx} \llbracket \Gamma, \mathsf{let}\ x = \mathsf{domAPI.appendChild}(id_t, id_d, id_n, node_1, \ell_1)\ \mathsf{in}\ cmd \rrbracket_{id}, id)\\ \kappa = {\ell_1}^- \qquad t = (id_t, \_, \mathit{Docs} :: \mathit{Doc}, \cdots)\\ \mathit{Doc} = (id_d, \_, nodes, \_, \_) \qquad node \in_b nodes \qquad node = (id_n, (\cdots, \ell_{children}, \cdots), nodes_1, \cdots, \ell_n)\\ node_1 = (id_1, (\cdots, \ell_{parent}, \ell_{pre}, \cdot), \_, \ell_1) \qquad \nexists url \in attributes\\ \kappa \sqsubseteq {\ell_{children}}^* \qquad \kappa \sqsubseteq {\ell_{parent}}^* \qquad \kappa \sqsubseteq {\ell_{pre}}^* \qquad \ell_{succ} = \mathsf{nextSibleOf}(\mathsf{last}(nodes_1)) \qquad \kappa \sqsubseteq {\ell_{succ}}^*\\ \mathit{Doc}' = \mathit{Doc}[node \Leftarrow \mathsf{append}(node, node'_1)] \qquad t' = t[\mathit{Doc} \Leftarrow \mathit{Doc}'] \qquad \Sigma'_{ctx}(id) = \Sigma_{ctx}(id)[t \Leftarrow t']\end{array}}{\begin{array}{l}\Sigma_{ctx} \llbracket \Gamma, \mathsf{let}\ x = \mathsf{domAPI.appendChild}(id_t, id_d, id_n, node_1, \ell_1)\ \mathsf{in}\ cmd \rrbracket_{id}\\ \xrightarrow{\mathsf{call}(\mathsf{domAPI.appendChild}, (id_t, id_d, id_n, node_1, \ell_1), \mathsf{pointersOf}(node'_1), \kappa)} \Sigma'_{ctx} \llbracket \Gamma, cmd[\mathsf{pointersOf}(node'_1)/x] \rrbracket_{id}; \cdot\end{array}}$$

$$\frac{\begin{array}{l}\textsc{DomAppendChildren(b)}\\ \Sigma_{ctx}(id) = (\Psi, \mathit{Tabs} :: t_{\mathit{ctx}}(id), \cdots)\\ \ell = \mathsf{ctxOfId}(\Sigma_{ctx} \llbracket \Gamma, \mathsf{let}\ x = \mathsf{domAPI.appendChild}(id_t, id_d, id_n, node_1, \ell_1)\ \mathsf{in}\ cmd \rrbracket_{id}, id)\\ \kappa = {\ell_1}^- \qquad t = (id_t, \_, \mathit{Docs} :: \mathit{Doc}, \cdots)\\ \mathit{Doc} = (id_d, \_, nodes, \_, \_) \qquad node \in_b nodes \qquad node = (id_n, (\cdots, \ell_{children}, \cdots), nodes_1, \cdots, \ell_n)\\ node_1 = (id_1, (\cdots, \ell_{parent}, \ell_{pre}, \cdot), \_, \ell_1) \qquad url \in attributes \qquad \text{“}doc\text{''} \notin attributes\\ \psi_1 = \mathsf{readyToFetch}(\kappa, (id_t, id_d, id_n), url) \qquad \kappa \sqsubseteq {\ell_{children}}^* \qquad \kappa \sqsubseteq {\ell_{parent}}^* \qquad \kappa \sqsubseteq {\ell_{pre}}^*\\ \ell_{succ} = \mathsf{nextSibleOf}(\mathsf{last}(nodes_1)) \qquad \kappa \sqsubseteq {\ell_{succ}}^* \qquad \mathit{Doc}' = \mathit{Doc}[node \Leftarrow \mathsf{append}(node, node'_1)]\\ t' = t[\mathit{Doc} \Leftarrow \mathit{Doc}'] \qquad \Sigma'_{ctx}(id) = \Sigma_{ctx}(id)[\Psi \Leftarrow \Psi :: \psi_1][t \Leftarrow t']\end{array}}{\begin{array}{l}\Sigma_{ctx} \llbracket \Gamma, \mathsf{let}\ x = \mathsf{domAPI.appendChild}(id_t, id_d, id_n, node_1, \ell_1)\ \mathsf{in}\ cmd \rrbracket_{id}\\ \xrightarrow{\mathsf{call}(\mathsf{domAPI.appendChild}, (id_t, id_d, id_n, node_1, \ell_1), \mathsf{pointersOf}(node'_1), \kappa)} \Sigma'_{ctx} \llbracket \Gamma, cmd[\mathsf{pointersOf}(node'_1)/x] \rrbracket_{id}; \cdot\end{array}}$$

$$\frac{\begin{array}{l}\Sigma_{ctx}(id) = (\Psi, Tabs :: t_{\mathbf{ctx}}(id), \cdots) \\ \ell = \mathsf{ctxOfId}(\Sigma_{ctx} \rrbracket \Gamma, \mathsf{let}\ x = \mathsf{domAPI.appendChild}(id_t, id_d, id_n, node_1, \ell_1)\ \mathsf{in}\ cmd \rrbracket_{id}, id) \\ \kappa = {\ell_1}^{-} \qquad t = (id_t, \_, Docs :: Doc, \cdots) \qquad Doc = (id_d, \_, nodes, \_, \_) \\ node \in_b nodes \qquad node = (id_n, (\cdots, \ell_{children}, \cdots), nodes_1, \cdots, \ell_n) \\ node_1 = (id_1, (\cdots, \ell_{parent}, \ell_{pre}, \cdot), \_, \ell_1) \qquad url \in attributes \\ \text{“}doc'' \in attributes \qquad \psi_1 = \mathsf{readyToFetchDoc}(\kappa, id_t, id_n, id_1, url) \\ \kappa \sqsubseteq {\ell_{children}}^{*} \qquad \kappa \sqsubseteq {\ell_{parent}}^{*} \qquad \kappa \sqsubseteq {\ell_{pre}}^{*} \qquad \ell_{succ} = \mathsf{nextSibleOf}(\mathsf{last}(nodes_1)) \\ \kappa \sqsubseteq {\ell_{succ}}^{*} \qquad Doc' = Doc[node \Leftarrow \mathsf{append}(node, node_1')] \\ t' = t[Doc \Leftarrow Doc'] \qquad \Sigma'_{ctx}(id) = \Sigma_{ctx}(id)[\Psi \Leftarrow \Psi :: \psi_1][t \Leftarrow t'] \end{array}}{\begin{array}{l}\Sigma_{ctx} \rrbracket \Gamma, \mathsf{let}\ x = \mathsf{domAPI.appendChild}(id_t, id_d, id_n, node_1, \ell_1)\ \mathsf{in}\ cmd \rrbracket_{id} \\ \xrightarrow{\mathsf{call}(\mathsf{domAPI.appendChild}, (id_t, id_d, id_n, node_1, \ell_1), \mathsf{pointersOf}(node_1'), \kappa)} \Sigma'_{ctx} \rrbracket \Gamma, cmd[\mathsf{pointersOf}(node_1')/x] \rrbracket_{id}; \cdot\end{array}} \ \textsc{DomAppendChildren(c)}$$

DomRemoveChild A scrip can remove the child node from a node if all labels that need to be updated after the node is removed are lower than or equal to the scripts labe. That is, the labels of the parent, children, pre-sibling, and next-sibling allow it. Finally, we need to update the query set labels that are affected by the removal of the node. Function $\mathsf{updateQ}(Doc, \kappa, node_c)$ taints all the labels for queries that are affected by the removeal of $node_c$. For instance, if the node removed has a tag “p”, then the label for the query “p” needs to be updated.

$$\frac{\begin{array}{l}\Sigma_{ctx}(id) = (\Psi, Tabs :: t_{\mathbf{ctx}}(id), \cdots) \\ \ell = \mathsf{ctxOfId}(\Sigma_{ctx} \rrbracket \Gamma, \mathsf{let}\ x = \mathsf{domAPI.removeChild}(id_t, id_d, id_p, id_c)\ \mathsf{in}\ cmd \rrbracket_{id}, id) \\ \kappa = \ell^{-} \qquad t = (id_t, \_, Docs :: Doc, \cdots) \qquad Doc = (id_d, \_, nodes, \_, \_) \qquad node \in nodes \\ node = (id_p, m, nodes_1 :: node_1 :: node_c :: node_2 :: nodes_2, (\cdots, \ell_{children}, \cdots), \ell_p) \\ node_1 = (\cdots, (\cdots, \ell_{succ}, \cdots), \cdots) \qquad node_2 = (\cdots, (\cdots, \ell_{pre}, \cdots), \cdots) \\ \kappa \sqsubseteq {\ell_{children}}^{*} \qquad \kappa \sqsubseteq {\ell_{succ}}^{*} \qquad \kappa \sqsubseteq {\ell_{pre}}^{*} \\ node_1' = node_1[\ell_{succ} \Leftarrow \kappa \rhd_{tnt} \ell_{succ}] \qquad node_2' = node_1[\ell_{pre} \Leftarrow \kappa \rhd_{tnt} \ell_{pre}] \\ node' = (id_p, m, nodes_1 :: node_1' :: node_2' :: nodes_2, (\cdots, \kappa \rhd_{tnt} \ell_{children}, \cdots), \ell_p) \\ Doc' = \mathsf{updateQ}(Doc[node \Leftarrow node'], \kappa, node_c) \\ t' = t[Doc \Leftarrow Doc'] \qquad \Sigma_{ctx}(id)' = \Sigma_{ctx}(id)[t \Leftarrow t'] \end{array}}{\begin{array}{l}\Sigma_{ctx} \rrbracket \Gamma, \mathsf{let}\ x = \mathsf{domAPI.removeChild}(id_t, id_d, id_p, id_c)\ \mathsf{in}\ cmd \rrbracket_{id} \\ \xrightarrow{\mathsf{call}(\mathsf{domAPI.removeNode}, (id_t, id_d, id_p, id_c), \mathsf{pointersOf}(node), \kappa)} \Sigma'_{ctx} \rrbracket \Gamma, cmd[\mathsf{pointersOf}(node)/x] \rrbracket_{id}; \cdot\end{array}} \ \textsc{DomRemoveChild}$$

3.8.2 Rules for Accessing Bookmarks

We adopt multi-level bookmarks. There are 11 methods in the Bookmark API. The first six rules only query the bookmarks by using bookmarkQuery function. The last five rules update the bookmarks using bookmarkWrite function.

Query bookmarks First we define an auxiliary function selectBookmark to select a bookmark from the set of bookmarks $MBookmarks$ based on labels. The selection criteria include: 1) the label should be lower than κ; and 2) it is the latest updated one among the $MBookmarks$ with lower label than κ(or anything else depending on the implementation). The function selectBookmark returns a bookmark that satifies the above conditions. If no matching bookmarks can be found, the function returns NONE.

$$\frac{\begin{array}{l}MBookmark_1 = (bookmarks, \kappa_1) \qquad MBookmark_1 \in MBookmarks \qquad \kappa_1 \sqsubseteq \kappa \\ \forall MBookmark \in MBookmarks \text{ s.t. } labOf(MBookmark) \sqsubseteq \kappa \\ \quad \mathsf{laterUpdated}(MBookmark_1, MBookmark) = \mathsf{true}\end{array}}{bookmarks = \mathsf{selectBookmark}(MBookmarks, \kappa)} \ \textsc{SelectBookmark}$$

BookmarkGet: Get a set of bookmark items based on their IDs. Given $MBookmarks$ and κ, we first select a bookmark using selectBookmark, then, we query the returned bookmark. The qurey result is placed in an event that will eventually trigger the callback function for processing the query result.

$$\frac{\begin{array}{l}\Sigma = (\Psi, \cdots, MBookmarks, \cdots) \qquad \Psi = \Psi' :: \mathsf{chrome.bookmarks.get}(\kappa, id_{cb}, ids) \\ bookmarks = \mathsf{selectBookmark}(MBookmarks, \kappa) \\ result = \mathsf{bookmarkQuery}(bookmarks, \mathsf{chrome.bookmarks.get}, ids) \\ fresh(id_e) \qquad e = (id_e, id_{cb}, \mathsf{none}, result, \kappa) \qquad \Sigma' = \Sigma[\Psi \Leftarrow \Psi']\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{chrome.bookmarks.get}(\kappa, id_{cb}, ids)) = \Sigma'; e}\ \textsc{BookmarkGet}$$

The rules BookmarkGetChildren, BookmarkGetRecent, BookmarkGetTree, BookmarkGetSubTree and BookmarkSearch are similar to BookmarkGet.

BookmarkGetChildren: Given an ID, get a bookmark item's children.

$$\frac{\begin{array}{l}\Sigma = (\Psi, \cdots, MBookmarks, \cdots) \\ \Psi = \Psi' :: \mathsf{chrome.bookmarks.getChildren}(\kappa, id_{cb}, id) \\ bookmarks = \mathsf{selectBookmark}(MBookmarks, \kappa) \\ result = \mathsf{bookmarkQuery}(bookmarks, \mathsf{chrome.bookmarks.getChildren}, id) \\ fresh(id_e) \qquad e = (id_e, id_{cb}, \mathsf{none}, result, \kappa) \qquad \Sigma' = \Sigma[\Psi \Leftarrow \Psi']\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{chrome.bookmarks.getChildren}(\kappa, id_{cb}, id)) = \Sigma'; e}\ \textsc{BookmarkGetChildren}$$

BookmarkGetRecent: Retrieves the recently added bookmarks. Returns at most $number$ bookmark items.

$$\frac{\begin{array}{l}\Sigma = (\Psi, \cdots, MBookmarks, \cdots) \\ \Psi = \Psi' :: \mathsf{chrome.bookmarks.getRecent}(\kappa, id_{cb}, number) \\ bookmarks = \mathsf{selectBookmark}(MBookmarks, \kappa) \\ result = \mathsf{bookmarkQuery}(bookmarks, \mathsf{chrome.bookmarks.getRecent}, number) \\ fresh(id_e) \qquad e = (id_e, id_{cb}, \mathsf{none}, result, \kappa) \qquad \Sigma' = \Sigma[\Psi \Leftarrow \Psi']\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{chrome.bookmarks.getRecent}(\kappa, id_{cb}, number)) = \Sigma'; e}\ \textsc{BookmarkGetRecent}$$

BookmarkGetTree: Retrieves the entire bookmarks hierarchy.

$$\frac{\begin{array}{l}\Sigma = (\Psi, \cdots, MBookmarks, \cdots) \qquad \Psi = \Psi' :: \mathsf{chrome.bookmarks.getTree}(\kappa, id_{cb}) \\ bookmarks = \mathsf{selectBookmark}(MBookmarks, \kappa) \\ result = \mathsf{bookmarkQuery}(bookmarks, \mathsf{chrome.bookmarks.getTree}) \\ fresh(id_e) \qquad e = (id_e, id_{cb}, \mathsf{none}, result, \kappa) \qquad \Sigma' = \Sigma[\Psi \Leftarrow \Psi']\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{chrome.bookmarks.getTree}(\kappa, id_{cb})) = \Sigma'; e}\ \textsc{BookmarkGetTree}$$

BookmarkGetSubTree: Retrieves the subtrees of bookmark node with ID id.

$$\frac{\begin{array}{l}\Sigma = (\Psi, \cdots, MBookmarks, \cdots) \\ \Psi = \Psi' :: \mathsf{chrome.bookmarks.getSubTree}(\kappa, id_{cb}, id) \\ bookmarks = \mathsf{selectBookmark}(MBookmarks, \kappa) \\ result = \mathsf{bookmarkQuery}(bookmarks, \mathsf{chrome.bookmarks.getSubTree}, id) \\ fresh(id_e) \qquad e = (id_e, id_{cb}, \mathsf{none}, result, \kappa) \qquad \Sigma' = \Sigma[\Psi \Leftarrow \Psi']\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{chrome.bookmarks.getSubTree}(\kappa, id_{cb}, id)) = \Sigma'; e} \text{ BOOKMARKGETSUBTREE}$$

BOOKMARKSEARCH: Searches for BookmarkTreeNodes that match with a given query.

$$\frac{\begin{array}{l}\Sigma = (\Psi, \cdots, MBookmarks, \cdots) \qquad \Psi = \Psi' :: \mathsf{chrome.bookmarks.search}(\kappa, id_{cb}, query) \\ bookmarks = \mathsf{selectBookmark}(MBookmarks, \kappa) \\ result = \mathsf{bookmarkQuery}(bookmarks, \mathsf{chrome.bookmarks.search}, query) \\ fresh(id_e) \qquad e = (id_e, id_{cb}, \mathsf{none}, result, \kappa) \qquad \Sigma' = \Sigma[\Psi \Leftarrow \Psi']\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{chrome.bookmarks.search}(\kappa, id_{cb}, query)) = \Sigma'; e} \text{ BOOKMARKSEARCH}$$

Update bookmarks The APIs that update bookmarks will only update one of the bookmark trees in the mulit-level bookmarks. If API caller's simple label exactly matches one bookmark tree, then that bookmark tree will be updated. Otherwise, the selectBookmark will be called to select the closest one to be updated.

BOOKMARKCREATE(1): Add a new bookmark item under a directory with ID id_{parent}. $positionIndex$ is the new item's position index. $title$ is the bookmark item title. url is the URL. There is a copy of MBookmark $MBookmark$ in $MBookmarks$ with the same label as the caller's label. In this case, we directly update $MBookmark$. If $MBookmark$ does not contain a directory with ID id_{parent}, we set $result$ as void; else, $result$ contains $(id_{parent}, positionIndex, title, url)$.

$$\frac{\begin{array}{l}\Sigma = (\Psi, \cdots, MBookmarks, \cdots) \\ \Psi = \Psi' :: \mathsf{chrome.bookmarks.create}(\kappa, id_{cb}, (id_{parent}, positionIndex, title, url)) \\ MBookmarks = MBookmarks'' :: (bookmarks, \kappa) \\ (bookmarks', result) = \mathsf{bookmarkWrite}(bookmarks, \mathsf{chrome.bookmarks.create}, \\ \qquad\qquad (id_{parent}, positionIndex, title, url)) \\ fresh(id_e) \\ e = (id_e, id_{cb}, \mathsf{none}, result, \kappa) \qquad MBookmarks' = MBookmarks'' :: (bookmark', \kappa) \\ \Sigma' = \Sigma[\Psi \Leftarrow \Psi'][MBookmarks \Leftarrow MBookmarks']\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{chrome.bookmarks.create}(\kappa, id_{cb}, (id_{parent}, positionIndex, title, url))) = \Sigma'; e} \text{ BOOKMARKCREATE(1)}$$

BOOKMARKCREATE(2): There does not exist a $MBookmark$ with label κ. We select a MBookmark $MBookmark_s$ through selectBookmark$(MBookmarks, \kappa)$, and then generate a new MBookmark $MBookmark'_s$ from $MBookmark_s$ by replacing its label with κ. Then we add a new bookmark item to $MBookmark'_s$. The result of the write operation will be returned through $result$.

$$\frac{\begin{array}{l} \Sigma = (\Psi, \cdots, \mathit{MBookmarks}, \cdots) \\ \Psi = \Psi' :: \textsf{chrome.bookmarks.create}(\kappa, id_{cb}, (id_{parent}, \mathit{positionIndex}, \mathit{title}, \mathit{url})) \\ \nexists \mathit{MBookmark} \in \mathit{MBookmarks} \text{ s.t. } \mathit{labOf}(\mathit{MBookmark}) = \kappa \\ \mathit{bookmarks}_1 = \textsf{selectBookmark}(\mathit{MBookmarks}, \kappa) \\ (\mathit{bookmarks}'_1, \mathit{result}) = \textsf{bookmarkWrite}(\mathit{bookmarks}_1, \textsf{chrome.bookmarks.create}, \\ \qquad\qquad (id_{parent}, \mathit{positionIndex}, \mathit{title}, \mathit{url})) \\ \mathit{fresh}(id_e) \\ e = (id_e, id_{cb}, \textsf{none}, \mathit{result}, \kappa) \qquad \mathit{MBookmarks}' = \mathit{MBookmarks} :: (\mathit{bookmarks}'_1, \kappa) \\ \Sigma' = \Sigma[\Psi \Leftarrow \Psi'][\mathit{MBookmarks} \Leftarrow \mathit{MBookmarks}'] \end{array}}{\textsf{nextStateC}(\Sigma, \textsf{chrome.bookmarks.create}(\kappa, id_{cb}, (id_{parent}, \mathit{positionIndex}, \mathit{title}, \mathit{url}))) = \Sigma'; e} \text{ BOOKMARKCREATE}(2)$$

The rules BOOKMARKMOVE(1), BOOKMARKUPDATE(1), BOOKMARKREMOVE(1), BOOKMARKREMOVETREE(1) are similar to the rule BOOKMARKCREATE(1).

The rules BOOKMARKMOVE(2), BOOKMARKUPDATE(2), BOOKMARKREMOVE(2), BOOKMARKREMOVETREE(2) are similar to the rule BOOKMARKCREATE(2).

BOOKMARKMOVE: Moves the specified BookmarkTreeNode to the provided location.

$$\frac{\begin{array}{l} \Sigma = (\Psi, \cdots, \mathit{MBookmarks}, \cdots) \\ \Psi = \Psi' :: \textsf{chrome.bookmarks.move}(\kappa, id_{cb}, (id, id_{parent}, \mathit{positionIndex})) \\ \mathit{MBookmarks} = \mathit{MBookmarks}'' :: (\mathit{bookmarks}, \kappa) \\ (\mathit{bookmarks}', \mathit{result}) = \textsf{bookmarkWrite}(\mathit{bookmarks}, \textsf{chrome.bookmarks.move}, \\ \qquad\qquad (\kappa, id_{cb}, (id, id_{parent}, \mathit{positionIndex}))) \\ \mathit{fresh}(id_e) \\ e = (id_e, id_{cb}, \textsf{none}, \mathit{result}, \kappa) \qquad \mathit{MBookmarks}' = \mathit{MBookmarks}'' :: (\mathit{bookmark}', \kappa) \\ \Sigma' = \Sigma[\Psi \Leftarrow \Psi'][\mathit{MBookmarks} \Leftarrow \mathit{MBookmarks}'] \end{array}}{\textsf{nextStateC}(\Sigma, \textsf{chrome.bookmarks.move}(\kappa, id_{cb}, (id, id_{parent}, \mathit{positionIndex}))) = \Sigma'; e} \text{ BOOKMARKMOVE}(1)$$

$$\frac{\begin{array}{l} \Sigma = (\Psi, \cdots, \mathit{MBookmarks}, \cdots) \\ \Psi = \Psi' :: \textsf{chrome.bookmarks.move}(\kappa, id_{cb}, (id, id_{parent}, \mathit{positionIndex})) \\ \nexists \mathit{MBookmark} \in \mathit{MBookmarks} \text{ s.t. } \mathit{labOf}(\mathit{MBookmark}) = \kappa \\ \mathit{bookmarks}_1 = \textsf{selectBookmark}(\mathit{MBookmarks}, \kappa) \\ (\mathit{bookmarks}'_1, \mathit{result}) = \textsf{bookmarkWrite}(\mathit{bookmarks}_1, \textsf{chrome.bookmarks.move}, \\ \qquad\qquad (\kappa, id_{cb}, (id, id_{parent}, \mathit{positionIndex}))) \\ \mathit{fresh}(id_e) \\ e = (id_e, id_{cb}, \textsf{none}, \mathit{result}, \kappa) \qquad \mathit{MBookmarks}' = \mathit{MBookmarks} :: (\mathit{bookmarks}'_1, \kappa) \\ \Sigma' = \Sigma[\Psi \Leftarrow \Psi'][\mathit{MBookmarks} \Leftarrow \mathit{MBookmarks}'] \end{array}}{\textsf{nextStateC}(\Sigma, \textsf{chrome.bookmarks.move}(\kappa, id_{cb}, (id, id_{parent}, \mathit{positionIndex}))) = \Sigma'; e} \text{ BOOKMARKMOVE}(2)$$

BOOKMARKUPDATE: Updates the properties of a bookmark or folder. Specify only the properties that you want to change; unspecified properties will be left unchanged. Note: Currently, only 'title' and 'url' are supported.

$$\frac{\begin{array}{l}\Sigma = (\Psi, \cdots, \mathit{MBookmarks}, \cdots) \\ \Psi = \Psi' :: \textsf{chrome.bookmarks.update}(\kappa, \mathit{id}_{cb}, (\mathit{id}, \mathit{title}, \mathit{url})) \\ \mathit{MBookmarks} = \mathit{MBookmarks}'' :: (\mathit{bookmarks}, \kappa) \\ (\mathit{bookmarks}', \mathit{result}) = \textsf{bookmarkWrite}(\mathit{bookmarks}, \textsf{chrome.bookmarks.update}, \\ \qquad\qquad (\kappa, \mathit{id}_{cb}, (\mathit{id}, \mathit{title}, \mathit{url}))) \\ \mathit{fresh}(\mathit{id}_e) \\ e = (\mathit{id}_e, \mathit{id}_{cb}, \textsf{none}, \mathit{result}, \kappa) \qquad \mathit{MBookmarks}' = \mathit{MBookmarks}'' :: (\mathit{bookmark}', \kappa) \\ \Sigma' = \Sigma[\Psi \Leftarrow \Psi'][\mathit{MBookmarks} \Leftarrow \mathit{MBookmarks}'] \end{array}}{\textsf{nextStateC}(\Sigma, \textsf{chrome.bookmarks.update}(\kappa, \mathit{id}_{cb}, (\mathit{id}, \mathit{title}, \mathit{url}))) = \Sigma'; e} \text{ BOOKMARKUPDATE(1)}$$

$$\frac{\begin{array}{l}\Sigma = (\Psi, \cdots, \mathit{MBookmarks}, \cdots) \\ \Psi = \Psi' :: \textsf{chrome.bookmarks.update}(\kappa, \mathit{id}_{cb}, (\mathit{id}, \mathit{title}, \mathit{url})) \\ \nexists \mathit{MBookmark} \in \mathit{MBookmarks} \text{ s.t. } \mathit{labOf}(\mathit{MBookmark}) = \kappa \\ \mathit{bookmarks}_1 = \textsf{selectBookmark}(\mathit{MBookmarks}, \kappa) \\ (\mathit{bookmarks}_1', \mathit{result}) = \textsf{bookmarkWrite}(\mathit{bookmarks}_1, \textsf{chrome.bookmarks.update}, \\ \qquad\qquad (\kappa, \mathit{id}_{cb}, (\mathit{id}, \mathit{title}, \mathit{url}))) \\ \mathit{fresh}(\mathit{id}_e) \\ e = (\mathit{id}_e, \mathit{id}_{cb}, \textsf{none}, \mathit{result}, \kappa) \qquad \mathit{MBookmarks}' = \mathit{MBookmarks} :: (\mathit{bookmarks}_1', \kappa) \\ \Sigma' = \Sigma[\Psi \Leftarrow \Psi'][\mathit{MBookmarks} \Leftarrow \mathit{MBookmarks}'] \end{array}}{\textsf{nextStateC}(\Sigma, \textsf{chrome.bookmarks.update}(\kappa, \mathit{id}_{cb}, (\mathit{id}, \mathit{title}, \mathit{url}))) = \Sigma'; e} \text{ BOOKMARKUPDATE(2)}$$

The chrome.bookmarks.remove API removes a bookmark or an empty bookmark folder.

$$\frac{\begin{array}{l}\Sigma = (\Psi, \cdots, \mathit{MBookmarks}, \cdots) \qquad \Psi = \Psi' :: \textsf{chrome.bookmarks.remove}(\kappa, \mathit{id}_{cb}, \mathit{id}) \\ \mathit{MBookmarks} = \mathit{MBookmarks}'' :: (\mathit{bookmarks}, \kappa) \\ (\mathit{bookmarks}', \mathit{result}) = \textsf{bookmarkWrite}(\mathit{bookmarks}, \textsf{chrome.bookmarks.remove}(\kappa, \mathit{id}_{cb}, \mathit{id})) \\ \mathit{fresh}(\mathit{id}_e) \\ e = (\mathit{id}_e, \mathit{id}_{cb}, \textsf{none}, \mathit{result}, \kappa) \qquad \mathit{MBookmarks}' = \mathit{MBookmarks}'' :: (\mathit{bookmark}', \kappa) \\ \Sigma' = \Sigma[\Psi \Leftarrow \Psi'][\mathit{MBookmarks} \Leftarrow \mathit{MBookmarks}'] \end{array}}{\textsf{nextStateC}(\Sigma, \textsf{chrome.bookmarks.remove}(\kappa, \mathit{id}_{cb}, \mathit{id})) = \Sigma'; e} \text{ BOOKMARKREMOVE(1)}$$

$$\frac{\begin{array}{l}\Sigma = (\Psi, \cdots, \mathit{MBookmarks}, \cdots) \qquad \Psi = \Psi' :: \textsf{chrome.bookmarks.remove}(\kappa, \mathit{id}_{cb}, \mathit{id}) \\ \nexists \mathit{MBookmark} \in \mathit{MBookmarks} \text{ s.t. } \mathit{labOf}(\mathit{MBookmark}) = \kappa \\ \mathit{bookmarks}_1 = \textsf{selectBookmark}(\mathit{MBookmarks}, \kappa) \\ (\mathit{bookmarks}_1', \mathit{result}) = \textsf{bookmarkWrite}(\mathit{bookmarks}_1, \textsf{chrome.bookmarks.remove}(\kappa, \mathit{id}_{cb}, \mathit{id})) \\ \mathit{fresh}(\mathit{id}_e) \\ e = (\mathit{id}_e, \mathit{id}_{cb}, \textsf{none}, \mathit{result}, \kappa) \qquad \mathit{MBookmarks}' = \mathit{MBookmarks} :: (\mathit{bookmarks}_1', \kappa) \\ \Sigma' = \Sigma[\Psi \Leftarrow \Psi'][\mathit{MBookmarks} \Leftarrow \mathit{MBookmarks}'] \end{array}}{\textsf{nextStateC}(\Sigma, \textsf{chrome.bookmarks.remove}(\kappa, \mathit{id}_{cb}, \mathit{id})) = \Sigma'; e} \text{ BOOKMARKREMOVE(2)}$$

The chrome.bookmarks.removeTree API recursively removes a bookmark folder.

$$\frac{\begin{array}{l}\textsc{BookmarkRemoveTree}(1)\\ \mathit{\Sigma} = (\Psi, \cdots, \mathit{MBookmarks}, \cdots)\\ \Psi = \Psi' :: \textsf{chrome.bookmarks.removeTree}(\kappa, \mathit{id}_{cb}, \mathit{id}) \qquad \mathit{MBookmarks} = \mathit{MBookmarks}'' :: (\mathit{bookmarks}, \kappa)\\ (\mathit{bookmarks}', \mathit{result}) = \textsf{bookmarkWrite}(\mathit{bookmarks}, \textsf{chrome.bookmarks.removeTree}(\kappa, \mathit{id}_{cb}, \mathit{id}))\\ \mathit{fresh}(\mathit{id}_e) \qquad e = (\mathit{id}_e, \mathit{id}_{cb}, \textsf{none}, \mathit{result}, \kappa) \qquad \mathit{MBookmarks}' = \mathit{MBookmarks}'' :: (\mathit{bookmark}', \kappa)\\ \mathit{\Sigma}' = \mathit{\Sigma}[\Psi \Leftarrow \Psi'][\mathit{MBookmarks} \Leftarrow \mathit{MBookmarks}']\end{array}}{\textsf{nextStateC}(\mathit{\Sigma}, \textsf{chrome.bookmarks.removeTree}(\kappa, \mathit{id}_{cb}, \mathit{id})) = \mathit{\Sigma}'; e}$$

$$\frac{\begin{array}{l}\textsc{BookmarkRemoveTree}(2)\\ \mathit{\Sigma} = (\Psi, \cdots, \mathit{MBookmarks}, \cdots) \qquad \Psi = \Psi' :: \textsf{chrome.bookmarks.removeTree}(\kappa, \mathit{id}_{cb}, \mathit{id}))\\ \nexists \mathit{MBookmark} \in \mathit{MBookmarks} \text{ s.t. } \mathit{labOf}(\mathit{MBookmark}) = \kappa\\ \mathit{bookmarks}_1 = \textsf{selectBookmark}(\mathit{MBookmarks}, \kappa)\\ (\mathit{bookmarks}'_1, \mathit{result}) = \textsf{bookmarkWrite}(\mathit{bookmarks}_1, \textsf{chrome.bookmarks.removeTree}(\kappa, \mathit{id}_{cb}, \mathit{id}))\\ \mathit{fresh}(\mathit{id}_e) \qquad e = (\mathit{id}_e, \mathit{id}_{cb}, \textsf{none}, \mathit{result}, \kappa) \qquad \mathit{MBookmarks}' = \mathit{MBookmarks} :: (\mathit{bookmarks}'_1, \kappa)\\ \mathit{\Sigma}' = \mathit{\Sigma}[\Psi \Leftarrow \Psi'][\mathit{MBookmarks} \Leftarrow \mathit{MBookmarks}']\end{array}}{\textsf{nextStateC}(\mathit{\Sigma}, \textsf{chrome.bookmarks.removeTree}(\kappa, \mathit{id}_{cb}, \mathit{id})) = \mathit{\Sigma}'; e}$$

3.8.3 Rules for Accessing Cookies

The cookies can be set by server through the HTTP response header. In each web request, the browser sends all the cookies in the same domain to the web server, no matter whether the cookies were set by the server or client-site scripts. The httpOnly cookies cannot be accessed by client-side scripts including page scripts, extensions etc. A cookie for domain url always has a label $\kappa_c = \mathit{labFrom}(url)^-$.

CookieWebsiteSet(1): An web site updates an existing cookie item's value. We don't need label checking here because $\kappa = \mathit{labFrom}(url)^-$ and $\mathit{labFrom}(url)^-$ is the cookie's label.

$$\frac{\begin{array}{l}\mathit{\Sigma} = (\Psi, \cdots, \mathit{Cookies}, \cdots)\\ \Psi = \Psi' :: \textsf{website.cookieSet}(\kappa, \mathit{id}_{cb}, (\mathit{name}, \mathit{value}, \mathit{url})) \qquad \mathit{Cookie} \in \mathit{Cookies}\\ \mathit{Cookie} = (\mathit{name}, \mathit{value}_1, \mathit{url}, \kappa) \qquad \mathit{Cookie}' = \mathit{Cookie}[\mathit{value}_1 \Leftarrow \mathit{value}]\\ \mathit{Cookies}' = \mathit{Cookies}[\mathit{Cookie} \Leftarrow \mathit{Cookie}'] \qquad \mathit{\Sigma}' = \mathit{\Sigma}[\Psi \Leftarrow \Psi'][\mathit{Cookies} \Leftarrow \mathit{Cookies}']\\ \mathit{fresh}(\mathit{id}_e) \qquad e = (\mathit{id}_e, \mathit{id}_{cb}, \textsf{none}, \mathit{info}(\mathit{Cookie}'), \kappa)\end{array}}{\textsf{nextStateC}(\mathit{\Sigma}, \textsf{website.cookieSet}(\kappa, \mathit{id}_{cb}, (\mathit{name}, \mathit{value}, \mathit{url}))) = \mathit{\Sigma}'; e} \textsc{CookieWebsiteSet}(1)$$

CookieWebsiteSet(2): An web site sets a new cookie item.

$$\frac{\begin{array}{l}\mathit{\Sigma} = (\Psi, \cdots, \mathit{Cookies}, \cdots) \qquad \Psi = \Psi' :: \textsf{website.cookieSet}(\kappa, \mathit{id}_{cb}, (\mathit{name}, \mathit{value}, \mathit{url}))\\ \nexists \mathit{Cookie} \in \mathit{Cookies} \text{ s.t. } \mathit{Cookie} = (\mathit{name}, \cdot, \mathit{url}, \cdot) \qquad \mathit{Cookie} = (\mathit{name}, \mathit{value}, \mathit{url}, \kappa)\\ \mathit{Cookies}' = \mathit{Cookies} :: \mathit{Cookie} \qquad \mathit{\Sigma}' = \mathit{\Sigma}[\Psi \Leftarrow \Psi'][\mathit{Cookies} \Leftarrow \mathit{Cookies}']\\ \mathit{fresh}(\mathit{id}_e) \qquad e = (\mathit{id}_e, \mathit{id}_{cb}, \textsf{none}, \mathit{info}(\mathit{Cookie}), \kappa)\end{array}}{\textsf{nextStateC}(\mathit{\Sigma}, \textsf{website.cookieSet}(\kappa, \mathit{id}_{cb}, (\mathit{name}, \mathit{value}, \mathit{url}))) = \mathit{\Sigma}'; e} \textsc{CookieWebsiteSet}(2)$$

CookieWebsiteGet(1): In each HTTP web request header, the cookies that belong to the same domain are attached. By default, $labFrom(url) \sqsubseteq \kappa$. There is no callback function for "get cookie" commands in HTTP headers. Seems this rule is useless.

$$\frac{\begin{array}{l}\Sigma = (\Psi, \cdots, Cookies, \cdots) \qquad \Psi = \Psi' :: \mathsf{website.cookieGet}(\kappa, url) \\ Cookies = Cookies_1 :: Cookies_2 \qquad \forall Cookie \in Cookies_1, Cookie = (\_, \_, url, \_) \\ \nexists Cookie \in Cookies_2 \text{ s.t. } Cookie = (\_, \_, url, \_) \qquad \Sigma' = \Sigma[\Psi \Leftarrow \Psi'] \qquad e = \cdot\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{website.cookieGet}(\kappa, url)) = \Sigma'; e}\ \textsc{CookieWebsiteGet(1)}$$

CookieScriptSet(1): Asynchronous version. Successfully reset a cookie.

$$\frac{\begin{array}{l}\Sigma = (\Psi, \cdots, Cookies, \cdots) \\ \Psi = \Psi' :: \mathsf{domAPI.cookieSet}(\kappa, id_{cb}, (name, value, url)) \qquad Cookie \in Cookies \\ Cookie = (name, value_1, url, \kappa_c) \qquad \kappa \sqsubseteq \kappa_c \qquad Cookie' = Cookie[value_1 \Leftarrow value] \\ Cookies' = Cookies[Cookie \Leftarrow Cookie'] \qquad \Sigma' = \Sigma[\Psi \Leftarrow \Psi'][Cookies \Leftarrow Cookies'] \\ fresh(id_e) \qquad e = (id_e, id_{cb}, \mathsf{none}, info(Cookie'), \kappa_c)\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{domAPI.cookieSet}(\kappa, id_{cb}, (name, value, url))) = \Sigma'; e}\ \textsc{CookieScriptSet(1)}$$

CookieScriptSet(2): The label checking fails. The target cookie keeps unchanged.

$$\frac{\begin{array}{l}\Sigma = (\Psi, \cdots, Cookies, \cdots) \qquad \Psi = \Psi' :: \mathsf{domAPI.cookieSet}(\kappa, id_{cb}, (name, value, url)) \\ Cookie \in Cookies \qquad Cookie = (name, value', url, \kappa_c) \qquad \kappa \not\sqsubseteq \kappa_c \qquad \Sigma' = \Sigma[\Psi \Leftarrow \Psi']\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{domAPI.cookieSet}(\kappa, id_{cb}, (name, value, url))) = \Sigma'; \cdot}\ \textsc{CookieScriptSet(2)}$$

CookieScriptSet(3): A script creates a new cookie item.

$$\frac{\begin{array}{l}\Sigma = (\Psi, \cdots, Cookies, \cdots) \qquad \Psi = \Psi' :: \mathsf{domAPI.cookieSet}(\kappa, id_{cb}, (name, value, url)) \\ \nexists Cookie \in Cookies \text{ s.t. } Cookie = (name, \_, url, \_) \\ \kappa_c = labFrom(url) \qquad \kappa \sqsubseteq \kappa_c \qquad Cookie_{new} = (name, value, url, \kappa_c) \\ \Sigma' = \Sigma[\Psi \Leftarrow \Psi'][Cookies \Leftarrow Cookies :: Cookie_{new}] \\ fresh(id_e) \qquad e = (id_e, id_{cb}, \mathsf{none}, info(Cookie_{new}), \kappa_c)\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{domAPI.cookieSet}(\kappa, id_{cb}, (name, value, url))) = \Sigma'; e}\ \textsc{CookieScriptSet(3)}$$

CookieScriptSet(4): A script fails to create a new cookie item.

$$\frac{\begin{array}{l}\Sigma = (\Psi, \cdots, Cookies, \cdots) \qquad \Psi = \Psi' :: \mathsf{domAPI.cookieSet}(\kappa, id_{cb}, (name, value, url)) \\ \nexists Cookie \in Cookies \text{ s.t. } Cookie = (name, \_, url, \_) \\ \kappa_c = labFrom(url) \qquad \kappa \not\sqsubseteq \kappa_c \qquad \Sigma' = \Sigma[\Psi \Leftarrow \Psi']\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{domAPI.cookieSet}(\kappa, id_{cb}, (name, value, url))) = \Sigma'; \cdot}\ \textsc{CookieScriptSet(4)}$$

CookieScriptGet(1) Get a cookie.

$$\frac{\begin{array}{l}\Sigma = (\Psi, \cdots, \mathit{Cookies}, \cdots) \qquad \Psi = \Psi' :: \mathsf{domAPI.cookieGet}(\kappa, \mathit{id}_{cb}, (\mathit{name}, \mathit{url})) \\ \mathit{Cookie} \in \mathit{Cookies} \qquad \mathit{Cookie} = (\mathit{name}, \mathit{value}, \mathit{url}, \kappa_c) \\ \kappa_c \sqsubseteq \kappa \qquad \Sigma' = \Sigma[\Psi \Leftarrow \Psi'] \qquad \mathit{fresh}(\mathit{id}_e) \qquad e = (\mathit{id}_e, \mathit{id}_{cb}, \mathsf{none}, \mathit{Cookie}, \kappa)\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{domAPI.cookieGet}(\kappa, \mathit{id}_{cb}, (\mathit{name}, \mathit{url}))) = \Sigma'; e} \text{ COOKIESCRIPTGET(1)}$$

CookieScriptGet(2) Fail to get a cookie as the target cookie does not exist.

$$\frac{\begin{array}{l}\Sigma = (\Psi, \cdots, \mathit{Cookies}, \cdots) \qquad \Psi = \Psi' :: \mathsf{domAPI.cookieGet}(\kappa, \mathit{id}_{cb}, (\mathit{name}, \mathit{url})) \\ \nexists \mathit{Cookie} \in \mathit{Cookies} \text{ s.t. } \mathit{Cookie} = (\mathit{name}, \_, \mathit{url}, \_) \qquad \Sigma' = \Sigma[\Psi \Leftarrow \Psi']\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{domAPI.cookieGet}(\kappa, \mathit{id}_{cb}, (\mathit{name}, \mathit{url}))) = \Sigma'; \cdot} \text{ COOKIESCRIPTGET(2)}$$

CookieScriptGet(3) Fail to get a cookie due to the label checking.

$$\frac{\begin{array}{l}\Sigma = (\Psi, \cdots, \mathit{Cookies}, \cdots) \qquad \Psi = \Psi' :: \mathsf{domAPI.cookieGet}(\kappa, \mathit{id}_{cb}, (\mathit{name}, \mathit{url})) \\ \mathit{Cookie} \in \mathit{Cookies} \qquad \mathit{Cookie} = (\mathit{name}, \mathit{value}, \mathit{url}, \kappa_c) \qquad \kappa_c \not\sqsubseteq \kappa \qquad \Sigma' = \Sigma[\Psi \Leftarrow \Psi']\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{domAPI.cookieGet}(\kappa, \mathit{id}_{cb}, (\mathit{name}, \mathit{url}))) = \Sigma'; \cdot} \text{ COOKIESCRIPTGET(3)}$$

CookieCoreGet Extension core gets a cookie by (url, name, storeID). The browser return the cookie. Similar to COOKIESCRIPTGET rules.

CookieCoreSet Extension core set cookie with url, name. The rules are similar to COOKIESCRIPTSET rules.

CookieCoreGetAll Extension core gets a cookie with the parameters. The browser return the cookie, taint the label the extension core with the label of the cookies.

$$\frac{\begin{array}{l}\Sigma = (\Psi, \cdots, \mathit{Cookies}, \cdots) \qquad \Psi = \Psi' :: \mathsf{chrome.cookies.getAll}(\kappa, \mathit{id}_{cb}, \mathit{filter}) \\ \forall \mathit{Cookie} \in \mathit{Cookies}, \mathit{Cookie} = (\_, \_, \_, \kappa_c) \\ \quad \text{if } \mathsf{matchFilter}(\mathit{Cookie}, \mathit{filter}) = \mathsf{true} \wedge \kappa_c \sqsubseteq \kappa \\ \qquad \text{Add } \mathit{Cookie} \text{ to } \mathit{Cookies}_{results} \\ \Sigma' = \Sigma[\Psi \Leftarrow \Psi'] \qquad \mathit{fresh}(\mathit{id}_e) \qquad e = (\mathit{id}_e, \mathit{id}_{cb}, \mathsf{none}, \mathit{Cookies}_{results}, \kappa)\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{chrome.cookies.getAll}(\kappa, \mathit{id}_{cb}, \mathit{filter})) = \Sigma'; e} \text{ GOOKIECOREGETALL}$$

CookieCoreRemove(1) Extension core removes a cookie with $name$ and url. The callback function can access the removed cookie.

$$\frac{\begin{array}{l}\Sigma = (\Psi, \cdots, \mathit{Cookies}, \cdots) \qquad \Psi = \Psi' :: \mathsf{chrome.cookies.remove}(\kappa, \mathit{id}_{cb}, (\mathit{name}, \mathit{url})) \\ \mathit{Cookies} = \mathit{Cookies}' :: \mathit{Cookie} \\ \mathit{Cookie} = (\mathit{name}, \_, \mathit{url}, \kappa_c) \qquad \kappa \sqsubseteq \kappa_c \qquad \Sigma' = \Sigma[\Psi \Leftarrow \Psi'][\mathit{Cookies} \Leftarrow \mathit{Cookies}'] \\ \mathit{fresh}(\mathit{id}_e) \qquad e = (\mathit{id}_e, \mathit{id}_{cb}, \mathsf{none}, \mathit{Cookie}, \kappa_c)\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{chrome.cookies.remove}(\kappa, \mathit{id}_{cb}, (\mathit{name}, \mathit{url}))) = \Sigma'; e} \text{ COOKIECOREREMOVE(1)}$$

CookieCoreRemove(2) The target cookie does not exist.

$$\frac{\begin{array}{l}\Sigma = (\Psi, \cdots, \mathit{Cookies}, \cdots) \qquad \Psi = \Psi' :: \mathsf{chrome.cookies.remove}(\kappa, \mathit{id}_{cb}, (\mathit{name}, \mathit{url})) \\ \nexists \mathit{Cookie} \in \mathit{Cookies} \text{ s.t. } \mathit{Cookie} = (\mathit{name}, \_, \mathit{url}, \_) \qquad \Sigma' = \Sigma[\Psi \Leftarrow \Psi']\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{chrome.cookies.remove}(\kappa, \mathit{id}_{cb}, (\mathit{name}, \mathit{url}))) = \Sigma'; \cdot} \textsc{CookieCoreRemove(2)}$$

CookieCoreRemove(3) Extension core fails to remove a cookie with name and url due to the label checking.

$$\frac{\begin{array}{l}\Sigma = (\Psi, \cdots, \mathit{Cookies}, \cdots) \qquad \Psi = \Psi' :: \mathsf{chrome.cookies.remove}(\kappa, \mathit{id}_{cb}, (\mathit{name}, \mathit{url})) \\ \mathit{Cookies} = \mathit{Cookies}' :: \mathit{Cookie} \\ \mathit{Cookie} = (\mathit{name}, \_, \mathit{url}, \kappa_c) \qquad \kappa \not\sqsubseteq \kappa_c \qquad \Sigma' = \Sigma[\Psi \Leftarrow \Psi']\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{chrome.cookies.remove}(\kappa, \mathit{id}_{cb}, (\mathit{name}, \mathit{url}))) = \Sigma'; \cdot} \textsc{CookieCoreRemove(3)}$$

3.8.4 Rules for Accessing History

historySearch Searches the history for the last visit time of each page matching the query.

$$\frac{\begin{array}{l}\Sigma = (\Psi, \cdots, \mathit{histories}, \cdots) \\ \Psi = \Psi' :: \mathsf{chrome.history.search}(\kappa, \mathit{id}_{cb}, \mathit{query}) \qquad \mathit{histories} = \mathit{histories}_1 :: \mathit{histories}_2 \\ \forall \mathit{history} \in \mathit{histories}_1, \mathit{history} = (\mathit{id}, \mathit{url}, \mathit{name}, \mathit{visitTime}, \mathit{visitType}, \kappa_h) \\ \quad \mathsf{matchQuery}(\mathit{history}, \mathit{query}) = \mathsf{true} \\ \quad \kappa_h \sqsubseteq \kappa \\ \quad \nexists \mathit{history}' \in \mathit{histories}, \mathit{history}' = (\mathit{id}', \mathit{url}, \mathit{name}, \mathit{visitTime}', \mathit{visitType}', \kappa_h'), \mathit{history}' \neq \mathit{history} \\ \qquad \text{s.t. } \mathit{visitTime}' \text{ is later than } \mathit{visitTime} \wedge \kappa_h' \sqsubseteq \kappa \\ \forall \mathit{history} \in \mathit{histories}_2, \mathit{history} = (\mathit{id}, \mathit{url}, \mathit{name}, \mathit{visitTime}, \mathit{visitType}, \kappa_h) \\ \quad \mathsf{matchQuery}(\mathit{history}, \mathit{query}) = \mathsf{true} \\ \quad \kappa_h \sqsubseteq \kappa \\ \quad \exists \mathit{history}' \in \mathit{histories}_1, \mathit{history}' = (\mathit{id}', \mathit{url}, \mathit{name}, \mathit{visitTime}', \mathit{visitType}', \kappa_h'), \\ \qquad \text{s.t. } \mathit{visitTime}' \text{ is later than } \mathit{visitTime} \wedge \kappa_h' \sqsubseteq \kappa \\ \Sigma' = \Sigma[\Psi \Leftarrow \Psi'] \qquad \mathit{fresh}(\mathit{id}_e) \qquad e = (\mathit{id}_e, \mathit{id}_{cb}, \mathsf{none}, \mathit{histories}_1, \kappa)\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{chrome.history.search}(\kappa, \mathit{id}_{cb}, \mathit{query})) = \Sigma'; e} \textsc{historySearch}$$

If the user visit a page for n times, the browser will generate n history items with the same url and name. For each page, we put its latest history item into $\mathit{histories}_1$, the rests are in $\mathit{histories}_2$. Line 3-7 say that any item in $\mathit{histories}_1$ is fresher than any other record for the same page. Line

Line 8-12 are to ensure that $\mathit{histories}_1$ contains all pages's latest history record items.

historyGetVisits: Retrieves information about visits to a URL.

$$\frac{\begin{array}{l}\Sigma = (\Psi, \cdots, \mathit{histories}, \cdots) \\ \Psi = \Psi' :: \mathsf{chrome.history.getVisits}(\kappa, \mathit{id}_{cb}, \mathit{url}) \qquad \mathit{histories} = \mathit{histories}_1 :: \mathit{histories}_2 \\ \forall \mathit{history} \in \mathit{histories}_1 \\ \quad \mathit{history} = (\_, \mathit{url}, \_, \_, \_, \kappa_h) \\ \quad \kappa_h \sqsubseteq \kappa \\ \nexists \mathit{history} \in \mathit{histories}_2, \text{ s.t. } \mathit{history} = (\_, \mathit{url}, \_, \_, \_, \kappa_h) \wedge \kappa_h \sqsubseteq \kappa \\ \Sigma' = \Sigma[\Psi \Leftarrow \Psi'] \qquad \mathit{fresh}(\mathit{id}_e) \qquad e = (\mathit{id}_e, \mathit{id}_{cb}, \mathsf{none}, \mathit{histories}_1, \kappa)\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{chrome.history.getVisits}(\kappa, \mathit{id}_{cb}, \mathit{url})) = \Sigma'; e} \textsc{historyGetVisits}$$

historyAddUrl: Adds a URL to the history at the current time.

$$\dfrac{\begin{array}{l}\Sigma = (\Psi, \cdots, \mathit{histories}, \cdots) \\ \Psi = \Psi' :: \textsf{chrome.history.addUrl}(\kappa, \mathit{id}_{cb}, (\mathit{name}, \mathit{url}, \mathit{visitTime}, \mathit{visitType})) \\ \mathit{histories}' = \mathit{histories} :: \mathit{history} \\ \mathit{fresh}(\mathit{id}) \qquad \mathit{history} = (\mathit{id}, \mathit{url}, \mathit{name}, \mathit{visitTime}, \mathit{visitType}, \kappa) \\ \Sigma' = \Sigma[\Psi \Leftarrow \Psi'][\mathit{histories} \Leftarrow \mathit{histories}'] \qquad \mathit{fresh}(\mathit{id}_e) \qquad e = (\mathit{id}_e, \mathit{id}_{cb}, \textsf{none}, \mathit{history}, \kappa)\end{array}}{\textsf{nextStateC}(\Sigma, \textsf{chrome.history.addUrl}(\kappa, \mathit{id}_{cb}, (\mathit{name}, \mathit{url}, \mathit{visitTime}, \mathit{visitType}))) = \Sigma'; e} \ \textsc{historyAddUrl}$$

historyDeleteUrl: Removes all occurrences of the given URL from the history. We delete all the matching items which has a label higher than κ. In the callback event, we return void instead of the deleted high history items.

$$\dfrac{\begin{array}{l}\Sigma = (\Psi, \cdots, \mathit{histories}, \cdots) \\ \Psi = \Psi' :: \textsf{chrome.history.deleteUrl}(\kappa, \mathit{id}_{cb}, \mathit{url}) \qquad \mathit{histories} = \mathit{histories}_1 :: \mathit{histories}_2 \\ \forall \mathit{history} \in \mathit{histories}_1 \\ \quad \mathit{history} = (\_, \mathit{url}, \_, \_, \_, \kappa_h) \\ \quad \kappa \sqsubseteq \kappa_h \\ \nexists \mathit{history} \in \mathit{histories}_2, \text{ s.t. } \mathit{history} = (\_, \mathit{url}, \_, \_, \_, \kappa_h) \ \wedge \ \kappa \sqsubseteq \kappa_h \\ \Sigma' = \Sigma[\Psi \Leftarrow \Psi'][\mathit{histories} \Leftarrow \mathit{histories}_2] \\ \mathit{fresh}(\mathit{id}_e) \qquad e = (\mathit{id}_e, \mathit{id}_{cb}, \textsf{none}, \textsf{void}, \kappa)\end{array}}{\textsf{nextStateC}(\Sigma, \textsf{chrome.history.deleteUrl}(\kappa, \mathit{id}_{cb}, \mathit{url})) = \Sigma'; e} \ \textsc{historyDeleteUrl}$$

3.9 Asynchronous APIs Calls

Asynchronous calls are encoded using nextStateC function.

3.9.1 Web Request API Calls

$$\dfrac{\begin{array}{l}\Sigma = (\Psi, \cdots) \qquad \Psi = \Psi' :: \textsf{webRequestGet}(\kappa, \mathit{id}_{cb}, \mathit{info}, \mathit{url}) \\ \mathit{fresh}(\mathit{id}_e) \qquad e = (\mathit{id}_e, \textsf{webRequest.onBeforeRequest}, \textsf{none}, \mathit{info}, \kappa) \\ \psi' = \textsf{ws.beforeRequest}(\kappa, \mathit{id}_{cb}, \cdot, \mathit{id}_e, \mathit{info}, \mathit{url}) \qquad \Sigma' = \Sigma[\Psi \Leftarrow \Psi' :: \psi]\end{array}}{\textsf{nextStateC}(\Sigma, \textsf{webRequestGet}(\kappa, \mathit{id}_{cb}, \mathit{info}, \mathit{url})) = \Sigma'; e} \ \textsc{WebRequestGet}$$

3.9.2 Messaging

The sending message APIs are asynchronous. The argument of the callback function is the response from the recipient of the message. Therefore, the browser needs to keep track of pairs of sending and receiving event handlers. The return fields of events and event handlers are used for this purpose. The return field in an event indicates that the browser's state should change once the event handler that processes this event finishes. Rules RETURN-CORE, RETURN-CS, RETURN-PAGE generate such a browser state.

Messaging asynchronous API call example In general asynchronous API calls, the trigged events do not contain return channels. However, the events triggered by messaging asynchronous API calls might contain return channels. Here is an example, suppose a content script wants to send a message to an extension core, it calls the API chrome.runtime.sendMessage through FIREAPICALL-A-CS with a callback function to address the core's reply. The browser generates a state chrome.runtime.sendMessage(κ, id_{cb}, (id, id_{ext}, $\mathit{message}$)), where κ and id are the content script's simple label and ID respectively, id_{cb} is the callback handler's event type, id_{ext} is the target extension's

ID, and $message$ is the message. In CSTOCORE1, the $\mathsf{chrome.runtime.sendMessage}(\kappa, id_{cb}, (id, id_{ext}, message))$ state is processed by issuing an event
$e = (id_e, \mathsf{chrome.runtime.onMessage}, \mathsf{some}(\mathsf{chrome.runtime.sendMessage}, id_{cb}), (id, id_{ext}, message), \kappa)$. Then the state $\mathsf{chrome.runtime.sendMessage}(\kappa, id_{cb}, (id, id_{ext}, message))$ is replaced with the $\mathsf{chrome.runtime.sendMessage.waitResponse}(\kappa, id_{cb})$ state which indicates there is content script waiting for a reply to be handled by an event handler with event type id_{cb}. The event e contains a return channel: $\mathsf{some}(\mathsf{chrome.runtime.sendMessage}, id_{cb})$. With EVENTFIRE2, the event e is added to all the $\mathsf{chrome.runtime.onMessage}$ handlers in the target extension core. Each handler may return a reply to the content script. However, only the first reply will be accepted by the browser and then be passed to the content script. When a response is sent by the core (as modeled by RETURN-CORE), a new state $\mathsf{chrome.runtime.sendMessage.ResponseGenerated}(\kappa, id_{cb}, response)$ is generated and added into the browser. Now there are two states related to the $\mathsf{chrome.runtime.sendMessage}$ API call, namely $\mathsf{chrome.runtime.sendMessage.waitResponse}(\kappa, id_{cb})$ and $\mathsf{chrome.runtime.sendMessage.ResponseGenerated}(\kappa, id_{cb}, response)$. Then the browser knows a response has been sent by the target extension core, the internal transition rule CSTOCORE2 is applied, and a callback event $e = (id_e, id_{cb}, \mathsf{none}, response, \kappa)$ is issued. The callback events carries the core's response. Finally, the content script's callback handler receives the callback event and gets the core's response.

CSToCore Content script sends a message to extension core, it may also have a callback function to deal with the core's response. id is the message sender's ID.

$$\frac{\begin{array}{l}\Sigma = (\Psi, \cdots)\\ \Psi = \Psi' :: \mathsf{chrome.runtime.sendMessage}(\kappa, id_{cb}, (id, id_{ext}, message)) \qquad fresh(id_e)\\ e = (id_e, \mathsf{chrome.runtime.onMessage}, \mathsf{some}(\mathsf{chrome.runtime.sendMessage}, id_{cb}), (id, id_{ext}, message), \kappa)\\ \psi' = \mathsf{chrome.runtime.sendMessage.waitResponse}(\kappa, id_{cb}) \qquad \psi' \sim_{nb} e \qquad \Sigma' = \Sigma[\Psi \Leftarrow \Psi' :: \psi]\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{chrome.runtime.sendMessage}(\kappa, id_{cb}, (id, id_{ext}, message))) = \Sigma'; e}\ \textsc{CSToCore1}$$

After the receiving the response, the state regarding the messaging will be removed, and a callback event will be issued.

$$\frac{\begin{array}{l}\Sigma = (\Psi, \cdots)\\ \Psi = \Psi' :: \mathsf{chrome.runtime.sendMessage.waitResponse}(\kappa, id_{cb})\\ \qquad :: \mathsf{chrome.runtime.sendMessage.ResponseGenerated}(\kappa, id_{cb}, response)\\ fresh(id_e) \qquad e = (id_e, id_{cb}, \mathsf{none}, response, \kappa) \qquad \Sigma' = \Sigma[\Psi \Leftarrow \Psi']\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{chrome.runtime.sendMessage.ResponseGenerated}(\kappa, id_{cb}, response)) = \Sigma'; e}\ \textsc{CSToCore2}$$

CoreToCS: The rules are for one-time messaging from core to content scripts.In CORETOCS1, when a message event is sent from core, the state is changed to "waitResponse". The same, there is no callback event triggered in this rule. id_1 is the specific and unique event type for the callback event.

$$\frac{\begin{array}{l}\Sigma = (\Psi, \cdots)\\ \Psi = \Psi' :: \mathsf{chrome.tabs.sendMessage}(\kappa, id_{cb}, (id_{ext}, id_t, message)) \qquad fresh(id_e)\\ e = (id_e, \mathsf{chrome.runtime.onMessage}, \mathsf{some}(\mathsf{chrome.tabs.sendMessage}, id_{cb}), (id_{ext}, id_t, message), \kappa)\\ \psi' = \mathsf{chrome.tabs.sendMessage}(\kappa, id_{cb}, (id_{ext}, id_t, message)).\mathsf{WaitResponse}\\ \psi' \sim_{nb} e \qquad \Sigma' = \Sigma[\Psi \Leftarrow \Psi' :: \psi']\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{chrome.tabs.sendMessage}(\kappa, id_{cb}, (id_{ext}, id_t, message))) = \Sigma'; e}\ \textsc{CoreToCS1}$$

In CORETOCS2, the content script event handlers for the chrome.runtime.onMessage event have return a response(or none). And upon getting the response, the browser change the state to “responseGenerated”.

$$\frac{\begin{array}{l}\mathit{\Sigma} = (\Psi, \cdots) \\ \Psi = \Psi' :: \mathsf{chrome.tabs.sendMessage.waitResponse}(\kappa, id_{cb}) \\ \qquad :: \mathsf{chrome.tabs.sendMessage.ResponseGenerated}(\kappa, id_{cb}, \mathit{response}) \\ \mathit{fresh}(id_e) \qquad e = (id_e, id_{cb}, \mathsf{none}, \mathit{response}, \kappa) \qquad \mathit{\Sigma}' = \mathit{\Sigma}[\Psi \Leftarrow \Psi']\end{array}}{\mathsf{nextStateC}(\mathit{\Sigma}, \mathsf{chrome.tabs.sendMessage.ResponseGenerated}(\kappa, id_{cb}, \mathit{response})) = \mathit{\Sigma}'; e} \text{ CORETOCS2}$$

3.9.3 Chrome Tabs APIs

Create a new blank tabs When a user or an extension creates a new blank tab, a new tab t with fresh ID id_t is added to system state S. The tab creation triggers an event e. The parameters of chrome.tabs.create include windowId, index(optional integer), url, active(optional boolean), pinned (optional boolean), openerTabId(optional integer). The extensions create a new tab explicitly through chrome.tabs.create API. And we assume that, when receiving a user’s command, the handler UI also generates code chrome.tabs.create. id_w is window ID.

$$\frac{\begin{array}{l}\mathit{\Sigma} = (\Psi, \mathit{Tabs}, \mathit{Exts} :: \mathit{Ext}, \mathit{ExtCoreRs}, \cdots) \\ \Psi = \Psi' :: \mathsf{chrome.tabs.create}(\kappa, id_{cb}, (id_w, \mathit{positionIndex}, \cdot, \mathit{activeFlag}, \mathit{pinnedBoolean})) \\ \mathit{fresh}(id_t, id_{e1}, id_{e2}) \qquad t = (id_t, , \cdot, \mathit{EmptyUrl}, \cdot, \kappa^{FC}) \\ e_1 = (id_{e1}, \mathsf{tabs.onCreated}, \mathsf{none}, (id_t, \mathit{positionIndex}, id_w, \cdot, \mathit{EmptyUrl}), \kappa) \\ e_2 = (id_{e2}, id_{cb}, \mathsf{none}, \mathsf{getInfo}(t), \kappa)\end{array}}{\begin{array}{l}\mathsf{nextStateC}(\mathit{\Sigma}, \mathsf{chrome.tabs.create}(\kappa, id_{cb}, (id_w, \mathit{positionIndex}, \cdot, \mathit{activeFlag}, \mathit{pinnedBoolean}, \ell_t))) \\ = \mathit{\Sigma}[\Psi \Leftarrow \Psi'][\mathit{Tabs} \Leftarrow \mathit{Tabs} :: t], e_1 :: e_2\end{array}} \text{ NEWBLANKTAB}$$

When a new blank tab is created, we set the tab label the same as the creator’s label. Here κ is the label of the principal that wants to create the tab. κ_t is the label that the creator wants the tab to have. Based on API rules, it should be the case that $\kappa \Rightarrow \kappa_t$. If the creator is a user, by default, all extensions are allowed to access it. If the creator is an extension, by default, other extensions can not access it. id could be an extension ID or a user ID. The command can either be in extension context or user context. If the command comes from user context, id should be user, and the callback function cmd is set to $\cdot$.

Update a tab’s url We need to taint the label of the tab to prevent the case where the tab goes from H to L. If we overwrite the tab’s label, then the low execution may not be able to update the url.

$$\frac{\begin{array}{l}\mathit{\Sigma} = (\Psi, \mathit{Tabs} :: t, \cdots) \\ t = (id_t, \cdots, \ell_{tab}) \qquad \Psi = \Psi' :: \mathsf{chrome.tabs.update}(\kappa, id_{cb}, (id_t, \mathit{url}, \cdots)) \qquad \kappa \not\sqsubseteq {\ell_{tab}}^{*}\end{array}}{\mathsf{nextStateC}(\mathit{\Sigma}, \mathsf{chrome.tabs.update}(\kappa, id_{cb}, (id_t, \mathit{url}, \cdots))) = \mathit{\Sigma}[\Psi \Leftarrow \Psi'], \cdot} \text{ URLUPDATE1}$$

$$\frac{\begin{array}{l}\mathit{\Sigma} = (\Psi, \mathit{Tabs} :: t, \cdots) \qquad t = (id_t, \cdots, \ell_{tab}) \\ \Psi = \Psi' :: \mathsf{chrome.tabs.update}(\kappa, id_{cb}, (id_t, \mathit{url}, \cdots)) \qquad \kappa \sqsubseteq {\ell_{tab}}^{*} \\ \ell_1 = \kappa \rhd_{tnt} \ell_{tab} \qquad \psi' = \mathsf{readyToFetchDoc}(\kappa, id_t, \cdot, \cdot, \mathsf{url}) \qquad t' = t[\ell_{tab} \Leftarrow \ell_1] \\ \mathit{Tabs}' = \mathit{Tabs}[t \Leftarrow t'] \qquad \mathit{\Sigma} = \mathit{\Sigma}[\mathit{Tabs} \Leftarrow \mathit{Tabs}'][\Psi \Leftarrow \Psi' :: \psi'] \qquad \mathit{fresh}(id_1, id_2) \\ e_1 = (id_1, \mathsf{tabs.onUpdated}, \mathsf{none}, (id_t, \mathit{info}_1), {\ell_1}^{-}) \qquad e_2 = (id_2, id_{cb}, \mathsf{none}, \mathsf{getInfo}(t'), {\ell_1}^{-})\end{array}}{\mathsf{nextStateC}(\mathit{\Sigma}, \mathsf{chrome.tabs.update}(\kappa, id_{cb}, (id_t, \mathit{url}, \cdots))) = \mathit{\Sigma}'; e_1 :: e_2} \text{ URLUPDATE2}$$

Create a blank tab with url: Here, we allow the new tab to float to top secret.

$$\frac{\begin{array}{l} \mathit{fresh}(id_t, id_d, id_{e1}, id_{e2}) \qquad \Sigma = (\Psi, \mathit{Tabs}, \cdots) \\ \Psi = \Psi' :: \mathsf{chrome.tabs.create}(\kappa, id_{cb}, (id_w, \mathit{positionIndex}, \mathit{url}, \mathit{activeFlag}, \mathit{pinnedBoolean}, id_{parentTab})) \\ t = (id_t, \cdot, \mathit{url}, \cdot, \kappa^{FC}) \\ \psi' = \mathsf{readyToFetchDoc}(\kappa, id_t, \cdot, \cdot, \mathit{url}) \qquad \Sigma' = \Sigma[\Psi \Leftarrow \Psi' :: \psi][\mathit{Tabs} \Leftarrow \mathit{Tabs} :: t] \\ \mathcal{E} = (id_{e1}, \mathsf{tabs.onCreated}, \mathsf{none}, \mathit{info}_1, \kappa) :: (id_{e2}, id_{cb}, \mathsf{none}, \mathsf{getInfo}(t), \kappa) \end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{chrome.tabs.create}(\kappa, id_{cb}, (id_w, \mathit{positionIndex}, \mathit{url}, \mathit{activeFlag}, \mathit{pinnedBoolean}, id_{parentTab}))) = \Sigma', \mathcal{E}} \ \textsc{NewTab}$$

3.9.4 Script Injection

When a content script from an extension is ready to be injected, a new label for the runtime copy of the script is computed from the label associated with the state that the script is ready to be inject (κ) and the policy of the extension ($\mathit{extPolicy}(id_{ext})$). The label computation function essentially taints the policy label with κ ($\kappa \uplus_M \mathit{extPolicy}(id_{ext})$).

$$\frac{\begin{array}{l} \Sigma = (\Psi, \mathit{Tabs} :: t, \cdots, \mathit{progInjCSs}, \cdots) \\ \Psi = \Psi' :: \mathsf{chrome.tabs.executeScript}(\kappa, id_t, id_{ext}, (\mathit{cmd}, \mathit{EventHandlers}), \mathit{index}, \mathit{runat}, \_) \\ t = (id_t, \mathit{Docs} :: \mathit{Doc}, \mathit{url}, \_, \ell_t) \qquad \mathit{fresh}(id_{cs}) \qquad \ell = \mathit{computeLabel}(\kappa, \mathit{extPolicy}(id_{ext})) \\ \mathit{ExtCS}_{new} = (id_{ext}, id_{cs}, id_t, \mathit{cmd}, \mathit{EventHandlers}, \cdot, \mathit{runat}, \mathit{index}, \ell) \\ \Sigma' = \Sigma[\mathit{progInjCSs} \Leftarrow \mathit{progInjCSs} :: \mathit{ExtCS}_{new}][\Psi \Leftarrow \Psi'] \end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{chrome.tabs.executeScript}(\kappa, id_t, id_{ext}, (\mathit{cmd}, \mathit{EventHandlers}), \mathit{index}, \mathit{runat}, \_)) = \Sigma'; \emptyset} \ \textsc{CSInjection}$$

3.9.5 Managing Extensions

Enable an extension An extension can enable, disable another extension/app(including itself), and uninstall an extension/APP, but it cannot install other extensions or apps. Here, the id in the browserstate is the unique id matching the callback function. A call to enable an extension is discarded if the extension is already active.

$$\frac{\begin{array}{l} \Sigma = (\Psi, \mathit{Tabs}, \mathit{Exts} :: \mathit{Ext}, \mathit{ExtCoreRs}, \cdots) \\ \Psi = \Psi' :: \mathsf{chrome.management.setEnabled}(\kappa, id_{cb}, (\mathit{string}, \mathit{id}, id_{ext}, \mathit{activeFlag})) \\ \mathit{Ext} = (id_{ext}, \mathit{ExtCore}, \cdots, \mathit{activeFlag}', \ell_1) \qquad \kappa \not\sqsubseteq {\ell_1}^* \end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{chrome.management.setEnabled}(\kappa, id_{cb}, (\mathit{string}, \mathit{id}, id_{ext}, \mathit{activeFlag}))) = \Sigma[\Psi \Leftarrow \Psi'], \cdot} \ \textsc{EnableDisableExt1(a)}$$

$$\frac{\begin{array}{l} \Sigma = (\Psi, \mathit{Tabs}, \mathit{Exts} :: \mathit{Ext}, \mathit{ExtCoreRs}, \cdots) \\ \Psi = \Psi' :: \mathsf{chrome.management.setEnabled}(\kappa, id_{cb}, (\mathit{string}, \mathit{id}, id_{ext}, \mathit{activeFlag})) \\ \mathit{Ext} = (id_{ext}, \mathit{ExtCore}, \cdots, \mathit{activeFlag}, \ell_1) \qquad \kappa \sqsubseteq {\ell_1}^* \\ \ell_2 = \kappa \rhd_{tnt} \ell_1 \qquad \mathit{fresh}(id_e) \qquad e_1 = (id_e, id_{cb}, \mathsf{none}, \mathsf{getInfo}(\mathit{Ext}), {\ell_2}^-) \end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{chrome.management.setEnabled}(\kappa, id_{cb}, (\mathit{string}, \mathit{id}, id_{ext}, \mathit{activeFlag}))) = \Sigma[\Psi \Leftarrow \Psi'], e_1} \ \textsc{EnableDisableExt1(b}$$

$$\frac{\begin{array}{l}\Sigma = (\Psi, \mathit{Tabs}, \mathit{Exts} :: \mathit{Ext}, \mathit{ExtCoreRs}, \cdots) \\ \Psi = \Psi' :: \mathsf{chrome.management.setEnabled}(\kappa, \mathit{id}_{cb}, (\mathit{string}, \mathit{id}, \mathit{id}_{ext}, \mathsf{active})) \\ \mathit{Ext} = (\mathit{id}_{ext}, \mathit{ExtCore}, \cdots, \mathit{activeFlag}, \ell_1) \\ \kappa \sqsubseteq {\ell_1}^{*} \qquad \mathit{activeFlag} = \mathsf{inactive} \qquad \ell_2 = \kappa \rhd_{tnt} \ell_1 \\ \mathit{ExtCoreR} = (\mathit{id}_{ext}, \mathit{ExtCore}, \ell_2) \qquad \mathit{Ext}' = \mathit{Ext}[\mathit{activeFlag} \Leftarrow \mathsf{active}][\ell_1 \Leftarrow \ell_2] \\ \Sigma' = \Sigma[\mathit{Ext} \Leftarrow \mathit{Ext}'][\mathit{ExtCoreRs} \Leftarrow \mathit{ExtCoreRs} :: \mathit{ExtCoreR}][\Psi \Leftarrow \Psi'] \\ \mathit{fresh}(\mathit{id}_{e1}, \mathit{id}_{e2}) \qquad e_1 = (\mathit{id}_{e1}, \mathit{id}_{cb}, \mathsf{none}, \mathsf{getInfo}(\mathit{Ext}'), {\ell_2}^{-}) \\ e_2 = (\mathit{id}_{e2}, \mathsf{management.onEnabled}, \mathsf{none}, \mathsf{getInfo}(\mathit{Ext}'), {\ell_2}^{-})\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{chrome.management.setEnabled}(\kappa, \mathit{id}_{cb}, (\mathit{string}, \mathit{id}, \mathit{id}_{ext}, \mathsf{active}))) = \Sigma', e_1 :: e_2} \ \textsc{enableExt2}$$

Disable an extension

$$\frac{\begin{array}{l}\Sigma = (\Psi, \mathit{Tabs}, \mathit{Exts} :: \mathit{Ext}, \mathit{ExtCoreRs}, \cdots) \\ \Psi = \Psi' :: \mathsf{chrome.management.setEnabled}(\kappa, \mathit{id}_{cb}, (\mathit{string}, \mathit{id}, \mathit{id}_{ext}, \mathsf{inactive})) \\ \mathit{Ext} = (\mathit{id}_{ext}, \mathit{ExtCore}, \cdots, \mathit{activeFlag}, \ell_1) \\ \kappa \sqsubseteq {\ell_1}^{*} \qquad \mathit{activeFlag} = \mathsf{active} \qquad \ell_2 = \kappa \rhd_{tnt} \ell_1 \qquad \mathit{ExtCoreRs} = \mathit{ExtCoreRs}' :: \mathit{ExtCoreR} \\ \mathit{ExtCoreR} = (\mathit{id}_{ext}, \cdots) \qquad \mathit{Ext}' = \mathit{Ext}[\mathit{activeFlag} \Leftarrow \mathsf{inactive}][\ell_1 \Leftarrow \ell_2] \\ \Sigma' = \Sigma[\mathit{Ext} \Leftarrow \mathit{Ext}'][\mathit{ExtCoreRs} \Leftarrow \mathit{ExtCoreRs}'][\Psi \Leftarrow \Psi'] \\ \mathit{fresh}(\mathit{id}_{e1}, \mathit{id}_{e2}) \qquad e_1 = (\mathit{id}_{e1}, \mathit{id}_{cb}, \mathsf{none}, \mathsf{getInfo}(\mathit{Ext}'), {\ell_2}^{-}) \\ e_2 = (\mathit{id}_{e2}, \mathsf{management.onDisabled}, \mathsf{none}, \mathsf{getInfo}(\mathit{Ext}'), {\ell_2}^{-})\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{chrome.management.setEnabled}(\kappa, \mathit{id}_{cb}, (\mathit{string}, \mathit{id}, \mathit{id}_{ext}, \mathsf{inactive}))) = \Sigma', e_1 :: e_2} \ \textsc{disableExt2}$$

The target extension does not exist.

$$\textsc{EnableDisableExt3}$$
$$\frac{\begin{array}{l}\Sigma = (\Psi, \mathit{Tabs}, \mathit{Exts}, \mathit{ExtCoreRs}, \cdots) \\ \Psi = \Psi' :: \mathsf{chrome.management.setEnabled}(\kappa, \mathit{id}_{cb}, (\mathit{string}, \mathit{id}, \mathit{id}_{ext}, \mathit{activeFlag})) \\ \nexists \mathit{Ext} \in \mathit{Exts} \qquad \mathit{Ext} = (\mathit{id}_{ext}, \cdots)\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{chrome.management.setEnabled}(\kappa, \mathit{id}_{cb}, (\mathit{string}, \mathit{id}, \mathit{id}_{ext}, \mathit{activeFlag}))) = \Sigma[\Psi \Leftarrow \Psi'], \cdot}$$

4 Noninterference

In this section, we introduce formal definitions of noninterference and state the main noninterference theorem.

4.1 Projections

We first define operations that remove elements from the state that an attacker cannot observe. We define the projection of relevant constructs given a simple label: $x \downarrow_\kappa$. The purpose of the projection is to remove information that cannot be read by the attacker (of label κ) from various constructs.

We define a function $\mathit{labOf}(x)$ to be the outmost label of the construct x.

Projection of actions:

$$\frac{{\ell_s}^{-} \sqsubseteq \kappa}{\mathit{API}(\ell_s, \cdots) \downarrow_\kappa = \mathit{API}(\ell_s, \cdots)} \ \text{API1} \qquad\qquad \frac{{\ell_s}^{-} \not\sqsubseteq \kappa}{\mathit{API}(\ell_s, \cdots) \downarrow_\kappa = \cdot} \ \text{API2}$$

Projection of events:

$$\frac{\mathit{labOf}(e) \sqsubseteq \kappa}{e \downarrow_\kappa = e}\ \textsc{event1} \qquad \frac{\mathit{labOf}(e) \not\sqsubseteq \kappa}{e \downarrow_\kappa = \cdot}\ \textsc{event2}$$

Projection of document scripts:

$$\frac{\mathit{DocCS} = (id_{ext}, id_{cs}, id_t, \Gamma, cmd, \mathit{EventHandlers}, \ell) \qquad \ell^- \sqsubseteq \kappa}{\mathit{DocCS} \downarrow_\kappa = (id_{ext}, id_{cs}, id_t, \Gamma, cmd, \mathit{EventHandlers} \downarrow_\kappa, \ell)}\ \text{DS1}$$

$$\frac{\mathit{labOf}(\mathit{DocCS})^- \not\sqsubseteq \kappa}{\mathit{DocCS} \downarrow_\kappa = \cdot}\ \text{DS2}$$

Projection of content scripts:

$$\frac{\mathit{labOf}(\mathit{ExtCS})^- \sqsubseteq \kappa}{\mathit{ExtCS} \downarrow_\kappa = \mathit{ExtCS}}\ \text{CS1} \qquad \frac{\mathit{labOf}(\mathit{ExtCS})^- \not\sqsubseteq \kappa}{\mathit{ExtCS} \downarrow_\kappa = \cdot}\ \text{CS2}$$

Projection of handlers

$$\frac{\mathit{EventHandler} = (id, \mathit{eventType}, x.cmd, \mathit{eventQueue}, cmd, id_e, \mathit{BlockingFlag}, \mathit{return})}{\mathit{EventHandler} \downarrow_\kappa = (id, \mathit{eventType}, x.cmd, \mathit{eventQueue} \downarrow_\kappa, cmd, id_e, \mathit{BlockingFlag}, \mathit{return})}\ \text{EH}$$

Projection of runtime extension cores:

$$\frac{\mathit{ExtCoreR} = (id_{ext}, (\Gamma, cmd, \mathit{EventHandlers}), \ell) \qquad \ell^- \sqsubseteq \kappa}{\mathit{ExtCoreR} \downarrow_\kappa = (id_{ext}, (\Gamma, cmd, \mathit{EventHandlers} \downarrow_\kappa), \ell)}\ \textsc{extCore1} \qquad \frac{\mathit{labOf}(\mathit{ExtCoreR})^- \not\sqsubseteq \kappa}{\mathit{ExtCoreR} \downarrow_\kappa = \cdot}\ \textsc{extCore2}$$

Projection of storage :

$$\frac{\mathit{labOf}(\mathit{Storage})^- \sqsubseteq \kappa}{\mathit{Storage} \downarrow_\kappa = \mathit{Storage}}\ \textsc{storage1} \qquad \frac{\mathit{labOf}(\mathit{Storage})^- \not\sqsubseteq \kappa}{\mathit{Storage} \downarrow_\kappa = \cdot}\ \textsc{storage2}$$

Projection of extensions:

$$\frac{\mathit{Ext} = (id_{ext}, \mathit{ExtCore}, \mathit{ExtCSs}, \mathit{Storage}, \mathit{activeFlag}, \ell) \qquad \ell^- \sqsubseteq \kappa}{\mathit{Ext} \downarrow_\kappa = (id_{ext}, \mathit{ExtCore}, \mathit{ExtCSs} \downarrow_\kappa, \mathit{Storage} \downarrow_\kappa, \mathit{activeFlag}, \ell)}\ \textsc{ext1}$$

$$\frac{\mathit{Ext} = (id_{ext}, \mathit{ExtCore}, \mathit{ExtCSs}, \mathit{Storage}, \mathit{activeFlag}, \ell) \qquad \ell^- \not\sqsubseteq \kappa}{\mathit{Ext} \downarrow_\kappa = (id_{ext}, \mathsf{void}, \mathit{ExtCSs} \downarrow_\kappa, \mathit{Storage} \downarrow_\kappa, \mathsf{void}, \mathsf{void})}\ \textsc{ext2}$$

Projection of a document node

$$\frac{\begin{array}{c} node = (id, (\cdots, \ell_{children}, \_, \ell_{succ}, \cdot), nodes, content, \ell) \\ \text{if } \ell^{-} \sqsubseteq \kappa \text{ then } c = content, l = \ell \text{ else } c = l = \mathsf{void} \\ \text{if } {\ell_{succ}}^{-} \sqsubseteq \kappa \text{ then } r = \mathsf{cnt} \text{ else } c = \mathsf{brk} \qquad {\ell_{children}}^{-} \sqsubseteq \kappa \qquad nodes \downarrow_{\kappa} = N, \Lambda \end{array}}{node \downarrow_{\kappa} = (id, attributes \downarrow_{\kappa}, N, c, l), \Lambda, r} \text{ NODE1}$$

$$\frac{\begin{array}{c} node = (id, (\cdots, \ell_{children}, \_, \ell_{succ}, \cdot), nodes, content, \ell) \\ \text{if } \ell^{-} \sqsubseteq \kappa \text{ then } c = content, l = \ell \text{ else } c = l = \mathsf{void} \\ \text{if } {\ell_{succ}}^{-} \sqsubseteq \kappa \text{ then } r = \mathsf{cnt} \text{ else } c = \mathsf{brk} \qquad {\ell_{children}}^{-} \not\sqsubseteq \kappa \qquad nodes \downarrow_{\kappa} = N, \Lambda \end{array}}{node \downarrow_{\kappa} = (id, attributes \downarrow_{\kappa}, \mathsf{void}, c, l), (N, \Lambda), r} \text{ NODE2} \qquad \frac{}{[] \downarrow_{\kappa} = [],} \text{ NODES1}$$

$$\frac{node \downarrow_{\kappa} = node', \Lambda, \mathsf{cnt} \qquad nodes \downarrow_{\kappa} = N :: N', \Lambda'}{node :: nodes \downarrow_{\kappa} = (node' :: N) :: N', (\Lambda, \Lambda')} \text{ NODES2}$$

$$\frac{node \downarrow_{\kappa} = node', \Lambda, \mathsf{brk} \qquad nodes \downarrow_{\kappa} = N, \Lambda'}{node :: nodes \downarrow_{\kappa} = [node'] :: N, (\Lambda, \Lambda')} \text{ NODES3}$$

Projection of a document page

$$\frac{Doc = (id_d, url, nodes, DocCSs, \ell) \qquad \ell^{-} \sqsubseteq \kappa}{Doc \downarrow_{\kappa} = (id_d, url, nodes \downarrow_{\kappa}, DocCSs \downarrow_{\kappa}, \ell)} \text{ DOC1} \qquad \frac{Doc = (id_d, url, nodes, DocCSs, \ell) \qquad \ell^{-} \not\sqsubseteq \kappa}{Doc \downarrow_{\kappa} = (id_d, \mathsf{void}, nodes \downarrow_{\kappa}, DocCSs \downarrow_{\kappa}, \mathsf{void})} \text{ DOC2}$$

Pruning page event handlers based on the result of document projection

$$\frac{EventHandler = (id, \cdots) \qquad id \in Docs}{\mathsf{prune}(EventHandler, Docs) = EventHandler} \text{ PAGEHANDLER1}$$

$$\frac{EventHandler = (id, \cdots) \qquad id \notin Docs}{\mathsf{prune}(EventHandler, Docs) = \cdot} \text{ PAGEHANDLER2}$$

Projection of browser states:

$$\frac{labOf(\psi) \sqsubseteq \kappa}{\psi \downarrow_{\kappa} = \psi} \text{ BROWSERSTATE1} \qquad \frac{labOf(\psi) \not\sqsubseteq \kappa}{\psi \downarrow_{\kappa} = \cdot} \text{ BROWSERSTATE2}$$

Projection of tabs:

$$\frac{\begin{array}{c} Tab = (id_t, Docs, url, EventHandlers, \ell) \\ \ell^{-} \sqsubseteq \kappa \qquad Docs' = Docs \downarrow_{\kappa} \qquad EventHandlers' = \mathsf{prune}(EventHandlers \downarrow_{\kappa}, Docs') \end{array}}{Tab \downarrow_{\kappa} = (id_t, Docs', url, EventHandlers', \ell)} \text{ TAB1}$$

$$\frac{\begin{array}{c} Tab = (id_t, Docs, url, EventHandlers, \ell) \\ \ell^{-} \not\sqsubseteq \kappa \qquad Docs' = Docs \downarrow_{\kappa} \qquad Docs' \neq \cdot \qquad EventHandlers' = \mathsf{prune}(EventHandlers \downarrow_{\kappa}, Docs') \end{array}}{Tab \downarrow_{\kappa} = (id_t, Docs', \mathsf{void}, EventHandlers', \mathsf{void})} \text{ TAB2}$$

$$\frac{Tab = (id_t, Docs, url, EventHandlers, \ell) \qquad \ell^{-} \not\sqsubseteq \kappa \qquad Docs \downarrow_{\kappa} = \cdot}{Tab \downarrow_{\kappa} = \cdot} \text{ TAB3}$$

Projection of MBookmarks:

$$\frac{\mathit{labOf}(\mathit{MBookmark}) \sqsubseteq \kappa}{\mathit{MBookmark} \downarrow_\kappa = \mathit{MBookmark}}\ \textsc{MBookmark1} \qquad \frac{\mathit{labOf}(\mathit{MBookmark}) \not\sqsubseteq \kappa}{\mathit{MBookmark} \downarrow_\kappa = \cdot}\ \textsc{MBookmark2}$$

Projection of cookies:

$$\frac{\mathit{labOf}(\mathit{Cookie}) \sqsubseteq \kappa}{\mathit{Cookie} \downarrow_\kappa = \mathit{Cookie}}\ \textsc{cookie1} \qquad \frac{\mathit{labOf}(\mathit{Cookie}) \not\sqsubseteq \kappa}{\mathit{Cookie} \downarrow_\kappa = \cdot}\ \textsc{cookie2}$$

Projection of histories:

$$\frac{\mathit{labOf}(\mathit{history}) \sqsubseteq \kappa}{\mathit{history} \downarrow_\kappa = \mathit{history}}\ \textsc{history1} \qquad \frac{\mathit{labOf}(\mathit{history}) \not\sqsubseteq \kappa}{\mathit{history} \downarrow_\kappa = \cdot}\ \textsc{history2}$$

4.2 Less than relation

Here, we define a binary relation that compares entities in two states. At a high-level, S_1 and S_2 are in this relation if S_1 and S_2 agrees on all the elements that an attacker can observe in S_1.

Less than Given two lists L_1 and L_2, we say L_1 is a sub-list of L_2, written $L_1 \leq L_2$ if $\forall i \in L_1, \exists j \in L_2$ and $i \leq j$. For any atomic (non-tuple) value x, $\mathsf{void} \leq x$, and $x \leq x$.

$$\frac{\mathit{EventHandlers}_1 \leq \mathit{EventHandlers}_2}{(\Gamma, \mathit{cmd}, \mathit{EventHandlers}_1) \leq (\Gamma, \mathit{cmd}, \mathit{EventHandlers}_2)}\ \textsc{extCore}$$

$$\frac{\mathit{ExtCore}_1 \leq \mathit{ExtCore}_2}{(\mathit{id}_{\mathit{ext}}, \mathit{ExtCore}_1, \ell) \leq (\mathit{id}_{\mathit{ext}}, \mathit{ExtCore}_2, \ell)}\ \textsc{extCoreR}$$

$$\frac{\mathit{EventHandlers}_1 \leq \mathit{EventHandlers}_2}{(\mathit{id}_{\mathit{ext}}, \mathit{id}_{\mathit{cs}}, \mathit{id}_t, \Gamma, \mathit{cmd}, \mathit{EventHandlers}_1, \ell) \leq (\mathit{id}_{\mathit{ext}}, \mathit{id}_{\mathit{cs}}, \mathit{id}_t, \Gamma, \mathit{cmd}, \mathit{EventHandlers}_2, \ell)}\ \textsc{docCS}$$

$$\frac{\mathit{url}_1 \leq \mathit{url}_2 \qquad \mathit{nodes}_1 \leq_{ns} \mathit{nodes}_2 \qquad \mathit{DocCSs}_1 \leq \mathit{DocCSs}_2 \qquad \ell_1 \leq \ell_2}{(\mathit{id}_d, \mathit{url}_1, \mathit{nodes}_1, \mathit{DocCSs}_1, \ell_1) \leq (\mathit{id}_d, \mathit{url}_2, \mathit{nodes}_2, \mathit{DocCSs}_2, \ell_2)}\ \textsc{Doc}$$

$$\frac{\mathit{Docs}_1 \leq \mathit{Docs}_2 \qquad \mathit{url}_1 \leq \mathit{url}_2 \qquad \mathit{EventHandlers}_1 \leq \mathit{EventHandlers}_2 \qquad \ell_1 \leq \ell_2}{(\mathit{id}_t, \mathit{Docs}_1, \mathit{url}_1, \mathit{EventHandlers}_1, \ell_1) \leq (\mathit{id}_t, \mathit{Docs}_2, \mathit{url}_2, \mathit{EventHandlers}_2, \ell_2)}\ \textsc{Tab}$$

$$\frac{}{\emptyset < N}\ \textsc{nodeSet1} \qquad \frac{N <_L N' : N3 \qquad N1 < N3, N2 : N4}{N, N1 < N', N2 : N4}\ \textsc{nodeSet2} \qquad \frac{nodes <_{nL} nodes' : N}{nodes <_L nodes' : N}\ \textsc{nodeList}$$

$$\frac{}{[] <_{nL} [] : \emptyset}\ \textsc{nodeList1} \qquad \frac{}{[] <_L N : N}\ \textsc{nodeList1'}$$

$$\frac{node <_n node' : N_1 \qquad nodes <_{nL} nodes' : N_2}{node :: nodes <_{nL} node' :: nodes' : N_1, N_2}\ \textsc{nodeList2}$$

$$\frac{node <_n node' : N_1 \qquad nodes <_L nodes' : N_2}{node :: nodes <_L node' :: nodes' : N_1, N_2}\ \textsc{nodeList2'}$$

$$\frac{nodes <_L nodes_2 : N}{nodes <_L nodes_1 @ nodes_2 : nodes_1, N}\ \textsc{nodeList3} \qquad \frac{node = (id, (), \mathsf{void}, \mathsf{void}, \mathsf{void})}{node :: nodes <_L nodes' : N}\ \textsc{nodeList4}$$

$$\frac{node = (\cdots, nodes', \cdots)}{[] <_L node :: nodes : nodes', nodes}\ \textsc{nodeList5} \qquad \frac{}{(id, (), \_, \mathsf{void}, \_) <_n (id, \_, nodes, \_) : nodes}\ \textsc{node1}$$

$$\frac{nodes <_{nL} nodes' : N}{(id, (), \mathsf{void}, nodes, \mathsf{void}) <_n (id, \_, nodes', \_) : N}\ \textsc{node2}$$

$$\frac{nodes <_{nL} nodes' : N}{(id, v_1, v_2, nodes, \ell) <_n (id, v_1, v_2, nodes', \ell) : N}\ \textsc{node3}$$

$$\frac{MBookmarks_1 = MBookmarks_2}{MBookmarks_1 \leq_{bkm} MBookmarks_2}\ \textsc{MBookmarks}$$

$$\frac{histories_1 = histories_2}{histories_1 \leq_{hstrs} histories_2}\ \textsc{Histories}$$

$$\frac{}{\cdot \leq_{B,e} \cdot}\ \textsc{extCoresB1} \qquad \frac{\begin{array}{c} ExtCoreR_1 \leq ExtCoreR_2 \qquad ExtCoreRs_1 \leq_{B,e} ExtCoreRs_2 \\ ExtCoreR_1 = (id_{ext}, (\_, \_, EventHandlers_1 :: eh_1), \_) \\ ExtCoreR_2 = (id_{ext}, (\_, \_, EventHandlers_2 :: eh_2), \_) \\ eh_1 = (eventTypeOf(e), \_, eventQueue_1, \_, \_, \mathsf{blocking}, \_) \\ eh_2 = (eventTypeOf(e), \_, eventQueue_2, \_, \_, \mathsf{blocking}, \_) \\ \text{if } e \notin eventQueue_1 \text{ then } e \notin eventQueue_2 \end{array}}{ExtCoreR_1 :: ExtCoreRs_1 \leq_{B,e} ExtCoreR_2 :: ExtCoreRs_2}\ \textsc{extCoresB2}$$

$$\frac{\begin{array}{c} ExtCoreRs_1 = ExtCoreRs_{b1} :: ExtCoreRs_{nb1} \\ ExtCoreRs_2 = ExtCoreRs_{b2} :: ExtCoreRs_{nb2} \qquad ExtCoreRs_{b1} \leq_{B,e} ExtCoreRs_{b2} \\ \text{every core in } ExtCoreRs_{b1} \text{ and } ExtCoreRs_{b2} \text{ contains a blocking event handler for } e \\ \text{no blocking event handlers for } e \text{ in } ExtCoreRs_{nb1} \text{ and } ExtCoreRs_{nb2} \end{array}}{ExtCoreRs_1 \leq_{c,e} ExtCoreRs_2}\ \textsc{extCores-E}$$

$$\frac{\text{for all blocking event } e, \mathit{ExtCoreRs}_1 \leq_{c,e} \mathit{ExtCoreRs}_2 \qquad \mathit{ExtCoreRs}_1 \leq \mathit{ExtCoreRs}_2}{\mathit{ExtCoreRs}_1 \leq_c \mathit{ExtCoreRs}_2}\ \textsc{extCores}$$

$$\frac{\begin{array}{l} e \text{ is a blocking event} \qquad e \sim_b \psi \\ \Psi_1 = \psi :: \Psi_{11} :: \Psi_{12} :: \Psi_{13} \qquad \Psi_{11} \text{ contains states of the form } \mathsf{ProcBlkEvtState}(\_,\_,e) \\ \Psi_{12} \text{ contains states of the form } \mathsf{DoneBlkEvtState}(\_, id, e, c) \\ \nexists\psi \in \Psi_{13}, \text{s.t. } \psi = \mathsf{ProcBlkEvtState}(\_,\_,e), \text{ or } \psi = \mathsf{DoneBlkEvtState}(\_,\_,e,\_) \\ \Psi_2 = \psi :: \Psi_{11} :: \Psi_{12} :: \Psi_{23} \\ \nexists\psi \in \Psi_{23}, \text{s.t. } \psi = \mathsf{ProcBlkEvtState}(\_,\_,e), \text{ or } \psi = \mathsf{DoneBlkEvtState}(\_, id, e, \_) \end{array}}{\Psi_1 \leq_{c,e} \Psi_2}\ \textsc{BrowserStates-E}$$

$$\frac{\text{for all blocking event } e, \Psi_1 \leq_{c,e} \Psi_2 \qquad \Psi_1 \leq \Psi_2}{\Psi_1 \leq_c \Psi_2}\ \textsc{browserStates}$$

$$\frac{\begin{array}{c} \Psi \leq_c \Psi' \\ \mathit{Tabs} \leq \mathit{Tabs}' \qquad \mathit{ExtCoreRs} \leq_c \mathit{ExtCoreRs}' \qquad \mathit{MBookmarks} \leq_{bkm} \mathit{MBookmarks}' \cdots\cdots \end{array}}{\begin{array}{c} \Psi, \mathit{Tabs}, \mathit{ExtCoreRs}, \mathit{progInjCSs}, \mathit{Exts}, \mathit{MCookies}, \mathit{MBookmarks}, \mathit{UI}, \mathit{sMode} \leq \\ \Psi', \mathit{Tabs}', \mathit{ExtCoreRs}', \mathit{progInjCSs}', \mathit{Exts}', \mathit{MCookies}', \mathit{MBookmarks}', \mathit{UI}, \mathit{sMode} \end{array}}\ \textsc{State}$$

Finally, we have an additionaly rule:

$$\frac{}{\mathsf{stuck} \leq \Sigma}\ \textsc{stuck}$$

4.3 Lemmas

We list several lemmas before presending our main noninterference theorem.

Definition 2 (Well-formed configuration)**.** *We define $wf(\Sigma)$ if all of the following hold:*

1. *$eh = (id, \cdots) \in Tab$ and $Tab \in \Sigma$, then node with id exists*
2. *$e = (id_e, \mathsf{DOM.scriptLoaded}, \mathsf{none}, (id_t, id_d, id), \kappa)$, $\kappa \not\sqsubseteq \ell_n$ where ℓ_n is the lable of node with ID id.*

Definition 3 (Matching component)**.** *We say that $comp_1$ and $comp_2$ are two matching states if the following holds:*

1. *$comp_1$ and $comp_2$ are components of the same type*
2. *one of the following hold:*
   (a) *$comp_1 \leq comp_2$ and one of the following holds*
      i. *$comp_1$ and $comp_2$ are run time extension cores and*
      ii. *$comp_1$ and $comp_2$ are injected content scripts ($DocCS$)*
      iii. *$comp_1$ and $comp_2$ are lists of injected content scripts ($DocCSs$)*
      iv. *$comp_1$ and $comp_2$ are tabs (Tab)*
   (b) *$comp_1 \leq_c comp_2$*

Lemma 1 (Browserstate event relation)**.** *If* $\psi \sim_b e$ *or* $\psi \sim_{nb} e$*, then* $\mathit{labOf}(\psi) = \mathit{labOf}(e)$*.*

Proof. By examining the definition of $\sim_b$ and $\sim_{nb}$. □

Lemma 2 (Event enqueue projection 1)**.**

1. *If* $\mathit{labOf}(e) \sqsubseteq \kappa$ *then*
   (a) $(\mathit{Tab} \lhd_Q e) \downarrow_\kappa = \mathit{Tab} \downarrow_\kappa \lhd_Q e$
   (b) $(\mathit{ExtCoreR} \lhd_Q e) \downarrow_\kappa = \mathit{ExtCoreR} \downarrow_\kappa \lhd_Q e$
2. *If* $\mathit{labOf}(e) \not\sqsubseteq \kappa$ *then*
   (a) $(\mathit{Tab} \lhd_Q e) \downarrow_\kappa = \mathit{Tab} \downarrow_\kappa$
   (b) $(\mathit{ExtCoreR} \lhd_Q e) \downarrow_\kappa = \mathit{ExtCoreR} \downarrow_\kappa$

Proof. By examining the definition of the enqueue operation. □

Lemma 3 (Event enqueue projection 2)**.**

1. *If* $\mathit{Tab}_1 \downarrow_\kappa \leq \mathit{Tab}_2 \downarrow_\kappa$ *then* $(\mathit{Tab}_1 \lhd_Q e) \downarrow_\kappa \leq (\mathit{Tab}_2 \lhd_Q e) \downarrow_\kappa$*.*
2. *If* $\mathit{ExtCoreR}_1 \downarrow_\kappa \leq \mathit{ExtCoreR}_2 \downarrow_\kappa$ *then* $(\mathit{ExtCoreR}_1 \lhd_Q e) \downarrow_\kappa \leq (\mathit{ExtCoreR}_2 \lhd_Q e) \downarrow_\kappa$*.*
3. *If* $\mathit{ExtCoreRs}_1 \downarrow_\kappa \leq_c \mathit{ExtCoreRs}_2 \downarrow_\kappa$ *then* $(\mathit{ExtCoreRs}_1 \lhd_Q e) \downarrow_\kappa \leq_c (\mathit{ExtCoreRs}_2 \lhd_Q e) \downarrow_\kappa$*.*

Lemma 4 (Node exists in Doc)**.** *If* $\mathit{node} = (\mathit{id}, \mathit{attributes}, \mathit{nodes}, \mathit{content}, \ell) \in \mathit{Docs}$ *and* $\ell^- \sqsubseteq \kappa$*, then* $\mathit{id} \in \mathit{Docs} \downarrow_\kappa$*.*

Lemma 5 (Node exists in Tab)**.** *If* $\mathit{node} = (\mathit{id}, \mathit{attributes}, \mathit{nodes}, \mathit{content}, \ell) \in \mathit{Tab}$ *and* $\mathit{Docs} \in \mathit{Tab}$*,* $\ell^- \sqsubseteq \kappa$*, then* $\mathit{id} \in \mathit{Docs} \downarrow_\kappa$*.*

Lemma 6 (Independent component update)**.** *If* $\mathit{comp}_1 \downarrow_\kappa$ *and* $\mathit{comp}_2 \downarrow_\kappa$ *are matching components,* $\mathit{comp}_a \downarrow_\kappa$ *and* $\mathit{comp}_b \downarrow_\kappa$ *are two pairs of matching components,* $\mathit{comp}'_a \downarrow_\kappa$ *and* $\mathit{comp}'_b \downarrow_\kappa$ *are two pairs of matching components, let* $\mathit{comp}'_1 = \mathit{comp}_1[\mathit{comp}_a \Leftarrow \mathit{comp}'_a]$*,* $\mathit{comp}'_2 = \mathit{comp}_2[\mathit{comp}_b \Leftarrow \mathit{comp}'_b]$*, then* $\mathit{comp}'_1 \downarrow_\kappa$ *and* $\mathit{comp}'_2 \downarrow_\kappa$ *are two pairs of matching components.*

Proof. Given the definitions of mathcing components, it is clear that the inner components comp_a and comp_b are projected independently of the outer components (comp_1 and comp_2). Therefore, updating the inner components with a matching pair preserves the less than relation. □

Lemma 7 (Event handler update)**.**

1. *If* $\mathit{DocCS}_1 = (\mathit{id}_{\mathit{ext}1}, \mathit{id}_{\mathit{cs}1}, \mathit{id}_{r1}, \Gamma_1, \mathit{cmd}_1, \mathit{EventHandlers}_1, \ell_1)$*,* $\mathit{DocCS}_2 = (\mathit{id}_{\mathit{ext}2}, \mathit{id}_{\mathit{cs}2}, \mathit{id}_{r2}, \Gamma_2, \mathit{cmd}_2, \mathit{EventHandlers}_2, \ell_2)$*,* $\mathit{DocCS}_1 \downarrow_\kappa \leq \mathit{DocCS}_2 \downarrow_\kappa$*,* ${\ell_1}^- \sqsubseteq \kappa$*,* $\mathit{EventHandlers}_1 = \mathit{EventHandlers}'_1 :: \mathit{eh}_1$*,* $\mathit{EventHandlers}_2 = \mathit{EventHandlers}'_2 :: \mathit{eh}_2$*,* $\mathit{eh}'_1 \downarrow_\kappa \leq \mathit{eh}'_2 \downarrow_\kappa$*, then* $\mathit{DocCS}_1[\mathit{eh}_1 \Leftarrow \mathit{eh}'_1] \downarrow_\kappa \leq \mathit{DocCS}_2[\mathit{eh}_2 \Leftarrow \mathit{eh}'_2] \downarrow_\kappa$*.*
2. *If* $\mathit{ExtCoreR}_1 = (\mathit{id}_{\mathit{ext}1}, (\Gamma_1, \mathit{cmd}_1, \mathit{EventHandlers}_1), \ell_1)$*,* $\mathit{ExtCoreR}_2 = (\mathit{id}_{\mathit{ext}2}, (\Gamma_2, \mathit{cmd}_2, \mathit{EventHandlers}_2), \ell_2)$*,* $\mathit{ExtCoreR}_1 \downarrow_\kappa \leq_c \mathit{ExtCoreR}_2 \downarrow_\kappa$*,* ${\ell_1}^- \sqsubseteq \kappa$*,* $\mathit{EventHandlers}_1 = \mathit{EventHandlers}'_1 :: \mathit{eh}_1$*,* $\mathit{EventHandlers}_2 = \mathit{EventHandlers}'_2 :: \mathit{eh}_2$*,* $\mathit{eh}'_1 \downarrow_\kappa \leq \mathit{eh}'_2 \downarrow_\kappa$*, for any blocking event* e*,* e *in the queue of* eh'_1 *iff* e *in the queue of* eh'_2 *then* $\mathit{ExtCoreR}_1[\mathit{eh}_1 \Leftarrow \mathit{eh}'_1] \downarrow_\kappa \leq_c \mathit{ExtCoreR}_2[\mathit{eh}_2 \Leftarrow \mathit{eh}'_2] \downarrow_\kappa$*.*

Proof. The projection of event handlers is independent of the rest of the elements, so swapping related event handlers on both sides mains the relation. □

Lemma 8 (Event handler update in page scripts). *If $Docs_1 \downarrow_\kappa \leq Docs_2 \downarrow_\kappa$, $node_1 \in Docs_1$ and $node_1 = (id_n, \cdots, \ell)$, $\ell^- \sqsubseteq \kappa$, $node_2 \in Docs_2$ and $node_2 = (id_n, \cdots, \ell)$, $node_1 \downarrow_\kappa \leq_n node_2 \downarrow_\kappa$, $\mathsf{prune}(EventHandlers_1 \downarrow_\kappa, Docs_1 \downarrow_\kappa) \leq \mathsf{prune}(EventHandlers_2 \downarrow_\kappa, Docs_2 \downarrow_\kappa)$, $EventHandlers_1 = EventHandlers'_1 :: eh_1$, $EventHandlers_2 = EventHandlers'_2 :: eh_2$, $eh'_1 \downarrow_\kappa \sqsubseteq eh'_2 \downarrow_\kappa$, $\ell'^- \sqsubseteq \kappa$, $node'_1 = node_1[\ell \Leftarrow \ell']$, $node'_2 = node_2[\ell \Leftarrow \ell']$, $node'_1 \downarrow_\kappa \leq_n node'_2 \downarrow_\kappa$ then $\mathsf{prune}(EventHandlers'_1 :: eh'_1 \downarrow_\kappa, Docs_1[node_1 \Leftarrow node'_1] \downarrow_\kappa) \leq \mathsf{prune}(EventHandlers'_2 :: eh'_2 \downarrow_\kappa, Docs_2[node_2 \Leftarrow node'_2] \downarrow_\kappa)$,*

Proof. Use Lemma 4. The updated node still exists after the projection. Further, the udpated event handlers are related. Therefore, the pruned event handlers are also related. □

Lemma 9 (Browserstate relation 1). *Given any blocking event e and $\Psi_1 \leq_c \Psi_2$, if*

1. *$\psi \sim_b e$*
2. *$\Psi_1 = \psi :: \Psi_{11} :: \Psi_{12} :: \Psi_{13}$,*
   Ψ_{11} contains states of the form $\mathsf{ProcBlkEvtState}(\_, \_, e)$,
   Ψ_{12} contains states of the form $\mathsf{DoneBlkEvtState}(\_, \_, e, \_)$,
   $\nexists \psi \in \Psi_{13}$ s.t. $\psi = \mathsf{ProcBlkEvtState}(\_, \_, e)$, or $\psi = \mathsf{DoneBlkEvtState}(\_, \_, e, \_)$,
3. *$\Psi_2 = \psi :: \Psi_{11} :: \Psi_{12} :: \Psi_{23}$,*
   $\nexists \psi \in \Psi_{23}$, s.t. $\psi = \mathsf{ProcBlkEvtState}(\_, \_, e)$, or $\psi = \mathsf{DoneBlkEvtState}(\_, \_, e, \_)$.

we have $\Psi_{13} \leq_c \Psi_{23}$.

Lemma 10 (Browserstate relation 2). *If $\Psi_1 \leq_c \Psi_2$, $\psi \notin \Psi_1$, $\Psi_1 :: \psi \leq_c \Psi_2 :: \psi$.*

Lemma 11 (Browserstate relation 3). *If $\Psi_1 \leq_c \Psi_2$, $\forall \psi \in \Psi_1$, for all blocking event e, $\psi \not\sim_b e$ imply $\Psi_1 \backslash \psi \leq_c \Psi_2 \backslash \psi$.*

Lemma 12 (Browserstate relation 4). *If $\Psi_1 \leq_c \Psi_2$, $\forall \psi \in \Psi_1$, for all blocking event e, $\psi \not\sim_b e$ and ψ is not a $\mathsf{DoneBlkEvtState}(\cdots)$ or $\mathsf{ProcBlkEvtState}(\cdots)$ related to e imply $\Psi_1 \backslash \psi \leq_c \Psi_2$.*

Lemma 13 (Next browserstate's label). *If $(\psi', e') = nextState(\psi, \mathsf{void})$, then $labOf(\psi) \sqsubseteq labOf(\psi') = labOf(e')$.*

Lemma 14 (nextStateC). *When the transitions are not* CSINJECTION, BEFOREFETCHEALLOWED, BEFOREFETCHDOCALLOWED, BEFOREFETCHDOCBLOCKED(1), BEFOREFETCHDOCBLOCKED(2), DOMAPPENDCHILDREN(1), *$\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$, $\psi \in \Sigma_1$, $\mathsf{nextStateC}(\Sigma_1, \psi) = \Sigma'_1, \mathcal{E}_1$ implies one of the following holds*

1. *$\mathsf{nextStateC}(\Sigma_2, \psi) = \Sigma'_2, \mathcal{E}_2$ and $\Sigma'_1 \downarrow_\kappa \leq \Sigma'_2 \downarrow_\kappa$, $\mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$.*
2. *$\Sigma'_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$, $\mathcal{E}_1 \downarrow_= \cdot$.*

Lemma 15 (Synchronous calls). *$\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$, $\Sigma_1 = \Sigma_{ctx}[\![\Gamma, API_s(\vec{e})]\!]_{id}$*

1. *if $\Sigma_1 \xrightarrow{\tau}_{API_s} \Sigma'_1; \mathcal{E}'_1$ then one of the following holds*
   (a) *$\Sigma_2 \xrightarrow{\tau}_{API_s} \Sigma'_2; \mathcal{E}'_2$ and $\Sigma'_1 \downarrow_\kappa \leq \Sigma'_2 \downarrow_\kappa$ and $\mathcal{E}'_1 \downarrow_\kappa \leq \mathcal{E}'_2 \downarrow_\kappa$*
   (b) *$\Sigma'_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$ and $\mathcal{E}'_1 \downarrow_\kappa = \cdot$*
2. *if $\Sigma_1 \xrightarrow{\alpha}_{API_s} \Sigma'_1; \mathcal{E}'_1$ and $\alpha \downarrow_\kappa = \cdot$ then $\Sigma'_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$ and $\mathcal{E}'_1 \downarrow_\kappa = \cdot$*
3. *if $\Sigma_1 \xrightarrow{\alpha}_{API_s} \Sigma'_1; \mathcal{E}'_1$ and $\alpha \downarrow_\kappa = \alpha$ then $\Sigma_2 \xrightarrow{\alpha}_{API_s} \Sigma'_2; \mathcal{E}'_2$, $\Sigma'_1 \downarrow_\kappa \leq \Sigma'_2 \downarrow_\kappa$ and $\mathcal{E}'_1 \downarrow_\kappa \leq \mathcal{E}'_2 \downarrow_\kappa$*

Lemma 16 (One step simulation). *$\Xi_1 \downarrow_\kappa \leq \Xi_2 \downarrow_\kappa$, $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$ and $\mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$*

1. *if $\Xi_1; \Sigma_1; \mathcal{E}_1 \xrightarrow{\tau} \Xi'_1; \Sigma'_1; \mathcal{E}'_1$ then one of the following holds*

(a) $\Xi_2; \Sigma_2; \mathcal{E}_2 \xrightarrow{\tau} \Xi_2'; \Sigma_2'; \mathcal{E}_2'$ *and* $\Xi_1' \downarrow_\kappa \leq \Xi_2' \downarrow_\kappa$, $\Sigma_1' \downarrow_\kappa \leq \Sigma_2' \downarrow_\kappa$, *and* $\mathcal{E}_1' \downarrow_\kappa \leq \mathcal{E}_2' \downarrow_\kappa$

(b) $\Xi_1' \downarrow_\kappa \leq \Xi_2' \downarrow_\kappa$, $\Sigma_1' \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$ *and* $\mathcal{E}_1' \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$

2. *if* $\Xi_1; \Sigma_1; \mathcal{E}_1 \xrightarrow{\alpha} \Xi_1'; \Sigma_1'; \mathcal{E}_1'$ *and* $\alpha \downarrow_\kappa = \cdot$ *then* $\Xi_1 \downarrow_\kappa \leq \Xi_2 \downarrow_\kappa$, $\Sigma_1' \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$ *and* $\mathcal{E}_1' \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$

3. *if* $\Xi_1; \Sigma_1; \mathcal{E}_1 \xrightarrow{\alpha} \Xi_1'; \Sigma_1'; \mathcal{E}_1'$ *and* $\alpha \downarrow_\kappa = \alpha$ *then* $\Xi_2; \Sigma_2; \mathcal{E}_2 \xrightarrow{\alpha} \Xi_2'; \Sigma_2'; \mathcal{E}_2'$, $\Xi_1' \downarrow_\kappa \leq \Xi_2' \downarrow_\kappa$, $\Sigma_1' \downarrow_\kappa \leq \Sigma_2' \downarrow_\kappa$ *and* $\mathcal{E}_1' \downarrow_\kappa \leq \mathcal{E}_2' \downarrow_\kappa$

Lemma 17 (Multi-step simulation)**.**
If $\Xi_1 \downarrow_\kappa \leq \Xi_2 \downarrow_\kappa$, $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$, $\mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$ *then* $\forall \rho$ *s.t.* $\Xi_1; \Sigma_1; \mathcal{E}_1 \overset{\rho}{\Longrightarrow}$ *and the transitions do not have declassification steps exists* ρ' *such that* $\rho \equiv_\kappa \rho'$ *and* $\Xi_2; \Sigma_2; \mathcal{E}_2 \overset{\rho'}{\Longrightarrow}$

Proof. By induction on the number of transition steps, in the inductive case, Use Lemma 16. □

Theorem 18 (Noninterference)**.**
If $\Xi_1 \downarrow_\kappa = \Xi_2 \downarrow_\kappa$, $\Sigma_1 \downarrow_\kappa = \Sigma_2 \downarrow_\kappa$, $\mathcal{E}_1 \downarrow_\kappa = \mathcal{E}_2 \downarrow_\kappa$ *then*

- $\forall \rho$ *s.t.* $\Xi_1; \Sigma_1; \mathcal{E}_1 \overset{\rho}{\Longrightarrow}$ *and the transitions do not have declassification steps exists* ρ' *such that* $\rho \equiv_\kappa \rho'$ *and* $\Xi_2; \Sigma_2; \mathcal{E}_2 \overset{\rho'}{\Longrightarrow}$
- $\forall \rho$ *s.t.* $\Xi_2; \Sigma_2; \mathcal{E}_2 \overset{\rho}{\Longrightarrow}$ *and the transitions do not have declassification steps exists* ρ' *such that* $\rho \equiv_\kappa \rho'$ *and* $\Xi_1; \Sigma_1; \mathcal{E}_1 \overset{\rho'}{\Longrightarrow}$

Proof. Because the initial states are the same, so $\Xi_1 \downarrow_\kappa \leq \Xi_2 \downarrow_\kappa$, $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$, $\mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$ and $\Xi_2 \downarrow_\kappa \leq \Xi_1 \downarrow_\kappa$, $\Sigma_2 \downarrow_\kappa \leq \Sigma_1 \downarrow_\kappa$, $\mathcal{E}_2 \downarrow_\kappa \leq \mathcal{E}_1 \downarrow_\kappa$.

The conclusion follows from Lemma 17 directly. □

5 Proofs

5.1 Proof of Lemma 16

.

case: CONTEXT

There are three subcases: (1) the script is a page script handler, (2) the script is a content script and (3) the script is in the extension core.

Subcase: the script is a page script handler.

Let $\Sigma_1 = (\Psi_1, \mathit{Tabs}_1 :: \mathit{Tab}_1, \mathit{ExtCoreRs}_1, \mathit{progInjCSs}_1, \mathit{Exts}_1, \mathit{Cookies}_1, \mathit{MBookmarks}_1, \mathit{histories}_1, \mathit{UI}_1)$
$\Sigma_2 = (\Psi_2, \mathit{Tabs}_2, \mathit{ExtCoreRs}_2, \mathit{progInjCSs}_2, \mathit{Exts}_2, \mathit{Cookies}_2, \mathit{MBookmarks}_2, \mathit{histories}_2, \mathit{UI}_2)$

By assumptions

$\Sigma_1 = \Sigma_{ctx}\llbracket \Gamma, \mathit{cmd} \rrbracket_{id}$

$\mathit{Tab}_1 = \mathit{Tab}_{\mathsf{ctx}}\llbracket \Gamma, \mathit{cmd} \rrbracket_{id} = (\mathit{TabID}_1, \mathit{Docs}_1 :: \mathit{Doc}_{env}\llbracket\Gamma\rrbracket, \mathit{url}_1, \mathit{EventHandlers}_1 :: \mathit{EventHandler}_{\mathsf{ctx}}\llbracket \mathit{cmd} \rrbracket_{id}, \ell_1)$

and $\Sigma_1' = \Sigma_{ctx}\llbracket \Gamma', \mathit{cmd}' \rrbracket_{id}$, $\mathcal{E}_1' = \mathcal{E}_1$

Assume that the label of the node with ID id is ℓ_n: $\mathsf{ctxOfId}(\Sigma_1, id) = \ell_n$

sub-subcase i. $\ell_n^- \sqsubseteq \kappa$

By Lemma 5,

$id \in \mathit{Docs}_1 \downarrow_\kappa$, and exists $\mathit{node}_{env}\llbracket \Gamma \rrbracket_{id} \in \mathit{Doc}_{env} \downarrow_\kappa$

By definition of Prune,

(1) $\mathit{EventHandler}_{\mathsf{ctx}}\llbracket \mathit{cmd} \rrbracket_{id} \in \mathit{Tab}_1 \downarrow_\kappa$

By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

(2) $Tabs_2 = Tabs'_2 :: Tab_2$ and $EventHandler_{\textbf{ctx}}\llbracket\, cmd \,\rrbracket_{id} \in Tab_2 \downarrow_\kappa$

By definitions and (2),

(3) exists $Docs'_2 \in Tab_2 \downarrow_\kappa$ and exists $node'_{env}\llbracket\, \Gamma \,\rrbracket_{id} \in Docs'_2$ s.t. $node_{\textbf{env}}\llbracket\, \Gamma \,\rrbracket_{id} \leq_n node'_{env}\llbracket\, \Gamma \,\rrbracket_{id}$

By (2) and (3)

(4) $Tab_2 = (\,TabID_2, Docs_2\llbracket\Gamma\rrbracket, url_2, EventHandlers_2 :: EventHandler_{\textbf{ctx}}\llbracket\, cmd \,\rrbracket_{id}, \ell_2)$

Apply CONTEXT to Σ_2, let $\Sigma_2 = \Sigma_{ctx2}\llbracket\, \Gamma, cmd \,\rrbracket_{id}$

(5) $\Sigma_{ctx2}\llbracket\, \Gamma, cmd \,\rrbracket_{id}; \mathcal{E}_2 \xrightarrow{\tau} \Sigma_{ctx2}\llbracket\, \Gamma', cmd' \,\rrbracket_{id}; \mathcal{E}_2$

By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$, $\Sigma_1 = \Sigma_{ctx}\llbracket\, \Gamma, cmd \,\rrbracket_{id}$, $\Sigma_2 = \Sigma_{ctx2}\llbracket\, \Gamma, cmd \,\rrbracket_{id}$,
and CONTEXT does not change Σ_{ctx} and Σ_{ctx2}

(6) $\Sigma_{ctx}\llbracket\, \Gamma', cmd' \,\rrbracket_{id} \downarrow_\kappa \leq \Sigma_{ctx2}\llbracket\, \Gamma', cmd' \,\rrbracket_{id} \downarrow_\kappa$

By assumption

(7) $\mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$

(8) $\mathcal{E}'_1 \downarrow_\kappa = \mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$

sub-subcase ii. $\ell_n{}^{-} \not\sqsubseteq \kappa$

By definitions

(1) $id \notin Docs_1 \downarrow_\kappa$ and $EventHandler_{\textbf{ctx}}\llbracket\, cmd \,\rrbracket_{id} \notin Tab_1 \downarrow_\kappa$ and $node_{\textbf{env}}\llbracket\, \Gamma \,\rrbracket_{id} \notin Tab_1 \downarrow_\kappa$

By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$, $\Sigma_1 = \Sigma_{ctx}\llbracket\, \Gamma, cmd \,\rrbracket_{id}$, and CONTEXT does not change Σ_{ctx}

(2) $\Sigma_{ctx}\llbracket\, \Gamma', cmd' \,\rrbracket_{id} \downarrow_\kappa = \Sigma_{ctx}\llbracket\, \Gamma, cmd \,\rrbracket_{id} \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

By assumption

(3) $\mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$

Subcase: the script is a content script handler

Let $\Sigma_1 = (\Psi_1, Tabs_1 :: Tab_1, ExtCoreRs_1, progInjCSs_1, Exts_1, Cookies_1, MBookmarks_1, histories_1, UI_1)$
$\Sigma_2 = (\Psi_2, Tabs_2, ExtCoreRs_2, progInjCSs_2, Exts_2, Cookies_2, MBookmarks_2, histories_2, UI_2)$

By assumptions

$\Sigma_1 = \Sigma_{ctx}\llbracket\, \Gamma, cmd \,\rrbracket_{id}$

$Tab_1 = Tab_{\textbf{ctx}}\llbracket\, \Gamma, cmd \,\rrbracket_{id} = (\,TabID_1, Docs_1 :: Doc_{\textbf{ctx}}\llbracket\, \Gamma, cmd \,\rrbracket_{id}, url_1, EventHandlers_1, \ell_1)$

$Doc_{\textbf{ctx}}\llbracket\, \Gamma, cmd \,\rrbracket_{id} = (id_d, url, nodes, DocCSs :: DocCS_{\textbf{ctx}}\llbracket\, \Gamma, cmd \,\rrbracket_{id}, \ell_d)$

$DocCSs_{\textbf{ctx}}\llbracket\, \Gamma, cmd \,\rrbracket_{id} = (id_{ext}, id, id_t, \llbracket\Gamma\rrbracket, \llbracket cmd\rrbracket, EventHandlers, \ell)$
or $(id_{ext}, id, id_t, \llbracket\Gamma\rrbracket, cmd, EventHandlers :: EventHandler_{\textbf{ctx}}\llbracket cmd\rrbracket, \ell)$

sub-subcase i. $\ell^{-} \sqsubseteq \kappa$

By DS1,

$DocCS_{\textbf{ctx}}\llbracket\, \Gamma, cmd \,\rrbracket_{id} \downarrow_\kappa = (id_{ext}, id, id_t, \llbracket\Gamma\rrbracket, \llbracket cmd\rrbracket, EventHandlers \downarrow_\kappa, \ell)$
or $DocCS_{\textbf{ctx}}\llbracket\, \Gamma, cmd \,\rrbracket_{id} \downarrow_\kappa = (id_{ext}, id, id_t, \llbracket\Gamma\rrbracket, cmd, EventHandlers \downarrow_\kappa :: EventHandler_{\textbf{ctx}}\llbracket cmd\rrbracket \downarrow_\kappa, \ell)$

By definition

(1) $DocCS_{\textbf{ctx}}\llbracket\, \Gamma, cmd \,\rrbracket_{id} \downarrow_\kappa \in Tab_1 \downarrow_\kappa$

By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

(2) $Tabs_2 = Tabs'_2 :: Tab_2$ and $DocCS_{\textbf{ctx}}\llbracket\, \Gamma, cmd \,\rrbracket_{id} \downarrow_\kappa \in Tab_2 \downarrow_\kappa$

By definitions and (2)

(3) exists $DocCS_{\textbf{ctx}}\llbracket\, \Gamma, cmd \,\rrbracket_{id} \in DocCSs'$, $DocCSs' \in Doc'$, $Doc' \in Docs'$, $Docs'_2 \in Tab_2 \downarrow_\kappa$

By (2) and (3)

(4) $Tab_2 = (\,TabID_2, Docs' :: Doc'_{\textbf{ctx}}\llbracket\, \Gamma, cmd \,\rrbracket_{id}, url_2, EventHandlers_2, \ell_2)$

Apply CONTEXT to Σ_2, let $\Sigma_2 = \Sigma_{ctx2}\llbracket\, \Gamma, cmd \,\rrbracket_{id}$

(5) $\Sigma_{ctx2}\llbracket\, \Gamma, cmd \,\rrbracket_{id}; \mathcal{E}_2 \xrightarrow{\tau} \Sigma_{ctx2}\llbracket\, \Gamma', cmd' \,\rrbracket_{id}; \mathcal{E}_2$

By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$, $\Sigma_1 = \Sigma_{ctx}\llbracket\, \Gamma, cmd \,\rrbracket_{id}$, $\Sigma_2 = \Sigma_{ctx2}\llbracket\, \Gamma, cmd \,\rrbracket_{id}$,
and CONTEXT does not change Σ_{ctx} and Σ_{ctx2}

(6) $\Sigma_{ctx}\llbracket\,\Gamma', cmd'\,\rrbracket_{id}\downarrow_\kappa \leq \Sigma_{ctx2}\llbracket\,\Gamma', cmd'\,\rrbracket_{id}\downarrow_\kappa$

By assumption

(7) $\mathcal{E}_1\downarrow_\kappa \leq \mathcal{E}_2\downarrow_\kappa$

(8) $\mathcal{E}_1'\downarrow_\kappa = \mathcal{E}_1\downarrow_\kappa \leq \mathcal{E}_2\downarrow_\kappa$

sub-subcase ii. $\ell^- \not\sqsubseteq \kappa$

By DS2

(1) $DocCS_{ctx}\llbracket\,\Gamma, cmd\,\rrbracket_{id} \notin Tab_1\downarrow_\kappa$

By $\Sigma_1\downarrow_\kappa \leq \Sigma_2\downarrow_\kappa$, $\Sigma_1 = \Sigma_{ctx}\llbracket\,\Gamma, cmd\,\rrbracket_{id}$, and CONTEXT does not change Σ_{ctx}

(2) $\Sigma_{ctx}\llbracket\,\Gamma', cmd'\,\rrbracket_{id}\downarrow_\kappa = \Sigma_{ctx}\llbracket\,\Gamma, cmd\,\rrbracket_{id}\downarrow_\kappa \leq \Sigma_2\downarrow_\kappa$

By assumption

(3) $\mathcal{E}_1\downarrow_\kappa \leq \mathcal{E}_2\downarrow_\kappa$

Subcase: the script is in extension core

Let $\Sigma_1 = (\Psi_1, Tabs_1, ExtCoreRs_1 :: ExtCoreR, progInjCSs_1, Exts_1, Cookies_1, MBookmarks_1, histories_1, UI_1)$
$\Sigma_2 = (\Psi_2, Tabs_2, ExtCoreRs_2, progInjCSs_2, Exts_2, Cookies_2, MBookmarks_2, histories_2, UI_2)$

By assumptions

$\Sigma_1 = \Sigma_{ctx}\llbracket\,\Gamma, cmd\,\rrbracket_{id}$

$ExtCoreR = ExtCoreR_{ctx}\llbracket\,\Gamma, cmd\,\rrbracket_{id} = (id, (\llbracket\Gamma\rrbracket, \llbracket cmd\rrbracket, EventHandlers), \ell)$
or $(id, (\llbracket\Gamma\rrbracket, cmd, EventHandlers :: EventHandler_{ctx}\llbracket cmd\rrbracket), \ell)$

sub-subcase i. $\ell^- \sqsubseteq \kappa$

By EXTCORE1,

$ExtCoreR_{ctx}\llbracket\,\Gamma, cmd\,\rrbracket_{id}\downarrow_\kappa = (id, (\llbracket\Gamma\rrbracket, \llbracket cmd\rrbracket, EventHandlers\downarrow_\kappa), \ell)$

or $ExtCoreR_{ctx}\llbracket\,\Gamma, cmd\,\rrbracket_{id}\downarrow_\kappa = (id, (\llbracket\Gamma\rrbracket, cmd, EventHandlers\downarrow_\kappa :: EventHandler_{ctx}\llbracket cmd\rrbracket\downarrow_\kappa), \ell)$

By definition

(1) $ExtCoreR_{ctx}\llbracket\,\Gamma, cmd\,\rrbracket_{id}\downarrow_\kappa \in \Sigma_1\downarrow_\kappa$

By $\Sigma_1\downarrow_\kappa \leq \Sigma_2\downarrow_\kappa$

(2) $ExtCoreR_{ctx}\llbracket\,\Gamma, cmd\,\rrbracket_{id}\downarrow_\kappa \in ExtCoreRs_2\downarrow_\kappa$

By definitions and (2)

(3) exists $ExtCoreR'_{ctx}\llbracket\,\Gamma, cmd\,\rrbracket_{id}\downarrow_\kappa \in ExtCoreRs_2\downarrow_\kappa$
s.t. $ExtCoreR_{ctx}\llbracket\,\Gamma, cmd\,\rrbracket_{id}\downarrow_\kappa \leq ExtCoreR'_{ctx}\llbracket\,\Gamma, cmd\,\rrbracket_{id}\downarrow_\kappa$

let $\Sigma_2 = \Sigma_{ctx2}\llbracket\,\Gamma, cmd\,\rrbracket_{id}$, apply CONTEXT to Σ_2,

(4) $\Sigma_{ctx2}\llbracket\,\Gamma, cmd\,\rrbracket_{id}; \mathcal{E}_2 \xrightarrow{\tau} \Sigma_{ctx2}\llbracket\,\Gamma', cmd'\,\rrbracket_{id}; \mathcal{E}_2$

By $\Sigma_1\downarrow_\kappa \leq \Sigma_2\downarrow_\kappa$, $\Sigma_1 = \Sigma_{ctx}\llbracket\,\Gamma, cmd\,\rrbracket_{id}$, $\Sigma_2 = \Sigma_{ctx2}\llbracket\,\Gamma, cmd\,\rrbracket_{id}$,
and rule CONTEXT does not change Σ_{ctx} and Σ_{ctx2}

(5) $\Sigma_{ctx}\llbracket\,\Gamma', cmd'\,\rrbracket_{id}\downarrow_\kappa \leq \Sigma_{ctx2}\llbracket\,\Gamma', cmd'\,\rrbracket_{id}\downarrow_\kappa$

By assumption

(6) $\mathcal{E}_1\downarrow_\kappa \leq \mathcal{E}_2\downarrow_\kappa$

(7) $\mathcal{E}_1'\downarrow_\kappa = \mathcal{E}_1\downarrow_\kappa \leq \mathcal{E}_2\downarrow_\kappa$

sub-subcase ii. $\ell^- \not\sqsubseteq \kappa$

By EXTCORE2

(1) $ExtCoreR\downarrow_\kappa = \cdot$, $ExtCoreR_{ctx}\llbracket\,\Gamma, cmd\,\rrbracket_{id} \notin \Sigma_1\downarrow_\kappa$

By $\Sigma_1\downarrow_\kappa \leq \Sigma_2\downarrow_\kappa$ $\Sigma_1 = \Sigma_{ctx}\llbracket\,\Gamma, cmd\,\rrbracket_{id}$, and CONTEXT does not change Σ_{ctx}

(2) $\Sigma_{ctx}\llbracket\,\Gamma', cmd'\,\rrbracket_{id}\downarrow_\kappa = \Sigma_{ctx}\llbracket\,\Gamma, cmd\,\rrbracket_{id}\downarrow_\kappa \leq \Sigma_2\downarrow_\kappa$

By assumption

(3) $\mathcal{E}_1\downarrow_\kappa \leq \mathcal{E}_2\downarrow_\kappa$

case: RETURN-CORE

Let $\Sigma_1 = (\Psi_1, \mathit{Tabs}_1, \mathit{ExtCoreRs}_1, \mathit{progInjCSs}_1, \mathit{Exts}_1, \mathit{Cookies}_1, \mathit{MBookmarks}_1, \mathit{histories}_1, \mathit{UI}_1)$
$\Sigma_2 = (\Psi_2, \mathit{Tabs}_2, \mathit{ExtCoreRs}_2, \mathit{progInjCSs}_2, \mathit{Exts}_2, \mathit{Cookies}_2, \mathit{MBookmarks}_2, \mathit{histories}_2, \mathit{UI}_2)$

By assumptions

(A1) $\Sigma_1 = \Sigma_{ctx}\llbracket \Gamma, \mathsf{ret}\ v \rrbracket_{id}$

(A2) $\mathit{ExtCoreR}_{\mathbf{ctx}}\llbracket \Gamma, \mathsf{ret}\ v \rrbracket_{id} \in \mathit{ExtCoreRs}_1$

(A3) $\mathit{ExtCoreR} = \mathit{ExtCoreR}_{\mathbf{ctx}}\llbracket \Gamma, \mathsf{ret}\ v \rrbracket_{id} = (id, (\llbracket\Gamma\rrbracket, \mathit{cmd}, \mathit{EventHandlers} :: \mathit{EventHandler}_{ctx}\llbracket \mathsf{ret}\ v\rrbracket), \ell)$

(A4) $\mathit{EventHandler}_{ctx}\llbracket \mathsf{ret}\ v\rrbracket = (id_{eh}, \mathit{eventType}, x.\mathit{cmd}, \mathcal{E}, \llbracket \mathsf{ret}\ v\rrbracket, id_e, \mathsf{nonblocking}, \mathsf{some}(R, id_r))$

(A5) $\Sigma_1' = \Sigma_{ctx}'\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} = \Sigma_{ctx}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}[\Psi_1 \Leftarrow \Psi_1 :: \psi_1]$

(A6) $\psi_1 = \mathit{ayncAPIRetState}(R, \ell^-, id_r, v)$

(A7) $\mathcal{E}_1' = \mathcal{E}_1$

sub-subcase i. $\ell^- \sqsubseteq \kappa$

By EXTCORE1,

(1) $\mathit{ExtCoreR}_{\mathbf{ctx}}\llbracket \Gamma, \mathsf{ret}\ v \rrbracket_{id} \downarrow_\kappa = (id, (\llbracket\Gamma\rrbracket, \mathit{cmd}, \mathit{EventHandlers} \downarrow_\kappa :: \mathit{EventHandler}_{\mathbf{ctx}}\llbracket \mathsf{ret}\ v\rrbracket \downarrow_\kappa), \ell)$

(2) $\mathit{ExtCoreR}_{\mathbf{ctx}}\llbracket \Gamma, \mathsf{ret}\ v \rrbracket_{id} \downarrow_\kappa \in \Sigma_1 \downarrow_\kappa$

By $\mathit{ExtCoreRs}_1 \downarrow_\kappa \leq \mathit{ExtCoreRs}_2 \downarrow_\kappa$

(3) $\exists \mathit{ExtCoreR}_{\mathbf{ctx}}'\llbracket \Gamma, \mathsf{ret}\ v \rrbracket_{id} \in \mathit{ExtCoreRs}_2$

s.t. $\mathit{ExtCoreR}_{\mathbf{ctx}}\llbracket \Gamma, \mathsf{ret}\ v \rrbracket_{id} \downarrow_\kappa \leq \mathit{ExtCoreR}_{\mathbf{ctx}}'\llbracket \Gamma, \mathsf{ret}\ v \rrbracket_{id} \downarrow_\kappa$

$\mathit{ExtCoreR}_{\mathbf{ctx}}'\llbracket \Gamma, \mathsf{ret}\ v \rrbracket_{id} \downarrow_\kappa = (id, (\llbracket\Gamma\rrbracket, \mathit{cmd}, \mathit{EventHandlers}' \downarrow_\kappa :: \mathit{EventHandler}_{\mathbf{ctx}}'\llbracket \mathsf{ret}\ v\rrbracket \downarrow_\kappa), \ell)$

$\mathit{EventHandlers} \downarrow_\kappa \leq \mathit{EventHandlers}' \downarrow_\kappa$

$\mathit{EventHandler}_{ctx}'\llbracket \mathsf{ret}\ v\rrbracket = (id_{eh}, \mathit{eventType}, x.\mathit{cmd}, \mathcal{E}', \llbracket \mathsf{ret}\ v\rrbracket, id_e, \mathsf{nonblocking}, \mathsf{some}(R, id_r))$

$\mathcal{E} \downarrow_\kappa \leq \mathcal{E}' \downarrow_\kappa$

let $\Sigma_2 = \Sigma_{ctx2}\llbracket \Gamma, \mathsf{ret}\ v \rrbracket_{id}$, apply RETURN-CORE to Σ_2,

(4) $\Sigma_{ctx2}\llbracket \Gamma, \mathsf{ret}\ v \rrbracket_{id}; \mathcal{E}_2 \xrightarrow{\tau} \Sigma_{ctx2}'\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}; \mathcal{E}_2$

(5) $\Sigma_{ctx2}'(id) = \Sigma_{ctx2}(id)[\Psi_2 \Leftarrow \Psi_2 :: \psi_2]$

(6) $\psi_2 = \mathit{ayncAPIRetState}(R, \ell^-, id_r, v) = \psi_1$

By BROWSERSTATE1

(7) $\psi_1 \downarrow_\kappa = \psi_1$, $\psi_2 \downarrow_\kappa = \psi_2$,

By $\Psi_1 \downarrow_\kappa \leq_c \Psi_2 \downarrow_\kappa$, Lemma 10

(8) $(\Psi_1 :: \psi_1) \downarrow_\kappa \leq_c (\Psi_2 :: \psi_2) \downarrow_\kappa$

By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$, $\Sigma_1 = \Sigma_{ctx}\llbracket \Gamma, \mathsf{ret}\ v \rrbracket_{id}$, $\Sigma_2 = \Sigma_{ctx2}\llbracket \Gamma, \mathsf{ret}\ v \rrbracket_{id}$, (5), (6), and (7) and RETURN-CORE only change the browser state,

(9) $\Sigma_{ctx}'\llbracket \Gamma', \mathsf{skip} \rrbracket_{id} \downarrow_\kappa \leq \Sigma_{ctx2}'\llbracket \Gamma', \mathsf{skip} \rrbracket_{id} \downarrow_\kappa$

By assumption

(10) $\mathcal{E}_1' \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$

sub-subcase ii. $\ell^- \not\sqsubseteq \kappa$

By EXTCORE2

(1) $\mathit{ExtCoreR} \downarrow_\kappa = \cdot$, $\mathit{ExtCoreR}_{\mathbf{ctx}}\llbracket \Gamma, \mathsf{ret}\ v \rrbracket_{id} \notin \Sigma_1 \downarrow_\kappa$

By BROWSERSTATE2, EXTCORE2

(2) $\psi_1 \downarrow_\kappa = \cdot$, $\mathit{ExtCoreR}_{\mathbf{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} \notin \Sigma_1' \downarrow_\kappa$

By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$, $\Sigma_1 = \Sigma_{ctx}\llbracket \Gamma, \mathsf{ret}\ v \rrbracket_{id}$, and $\Sigma_1' = \Sigma_{ctx}'\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}$, (1) and (2)

(3) $\Sigma_1' \downarrow_\kappa = \Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

By assumption

(4) $\mathcal{E}_1' \downarrow_\kappa = \mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$

case: RETURN-CS, RETURN-PAGE, RETURN-N-CORE, RETURN-N-CS, RETURN-N-PAGE: the proofs are similar to the proofs for RETURN-CORE.

case: RETURN-CORE-BLOCKING

$$\frac{\begin{array}{l}\Sigma_{ctx}(id) = (\Psi, Tabs, ExtCoreRs_{\mathit{ctx}}(id), \cdots) \\ \Psi = \Psi' :: \mathsf{ProcBlkEvtState}(\ell^-, id_r, e) \qquad ExtCoreR_{\mathit{ctx}}(id) \in ExtCoreRs_{\mathit{ctx}}(id) \\ ExtCoreR_{\mathit{ctx}}(id) = (id, \_, EventHandlers :: EventHandler_{\mathit{ctx}}, \ell) \\ EventHandler_{\mathit{ctx}}(id) = (\cdots, \mathsf{blocking}, \mathsf{some}(e, id_r)) \\ \psi = \mathsf{DoneBlkEvtState}(\ell^-, id_r, e, v) \qquad \Sigma'_{ctx}(id) = \Sigma_{ctx}(id)[\Psi \Leftarrow \Psi :: \psi]\end{array}}{\Sigma_{ctx} \llbracket \Gamma, \mathsf{ret}\ v \rrbracket_{id}; \mathcal{E} \xrightarrow{\tau} \Sigma'_{ctx} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id}; \mathcal{E}} \text{ RETURN-CORE-BLOCKING}$$

Let $\Sigma_1 = (\Psi_1, Tabs_1, ExtCoreRs_1, progInjCSs_1, Exts_1, Cookies_1, MBookmarks_1, histories_1, UI_1)$,
$ExtCoreRs_1 = ExtCoreRs''_1 :: ExtCoreR_1$
$\Sigma_2 = (\Psi_2, Tabs_2, ExtCoreRs_2, progInjCSs_2, Exts_2, Cookies_2, MBookmarks_2, histories_2, UI_2)$

By assumptions

(A1) $\Sigma_1 = \Sigma_{ctx} \llbracket \Gamma, \mathsf{ret}\ v \rrbracket_{id}$

(A2) $ExtCoreR_1 = ExtCoreR_{\mathit{ctx}} \llbracket \Gamma, \mathsf{ret}\ v \rrbracket_{id} = (id, (\llbracket \Gamma \rrbracket, cmd, EventHandlers_1 :: EventHandler_{1ctx} \llbracket \mathsf{ret}\ v \rrbracket), \ell)$

(A3) $EventHandler_{ctx} \llbracket \mathsf{ret}\ v \rrbracket = (id_{eh}, eventType, x.cmd, \mathcal{E}, \llbracket \mathsf{ret}\ v \rrbracket, id_e, \mathsf{blocking}, \mathsf{some}(e, id_r))$

(A4) $\Psi_1 = \Psi''_1 :: \psi_{1a}, \psi_{1a} = \mathsf{ProcBlkEvtState}(\ell^-, id_r, e)$

(A5) $\psi_{1b} = \mathsf{DoneBlkEvtState}(\ell^-, id_r, e, v)$

(A6) $\Sigma'_1 = \Sigma'_{ctx} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id} = \Sigma_{ctx} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id}[\Psi_1 \Leftarrow \Psi_1 :: \psi_{1b}]$

(A7) $\mathcal{E}'_1 = \mathcal{E}_1$

sub-subcase i. $\ell^- \sqsubseteq \kappa$

By EXTCORE1,

(1) $ExtCoreR_{\mathit{ctx}} \llbracket \Gamma, \mathsf{ret}\ v \rrbracket_{id} \downarrow_\kappa = (id, (\llbracket \Gamma \rrbracket, cmd, EventHandlers_1 \downarrow_\kappa :: EventHandler_{1\mathit{ctx}} \llbracket \mathsf{ret}\ v \rrbracket \downarrow_\kappa), \ell)$

By definition

(2) $ExtCoreR_{\mathit{ctx}} \llbracket \Gamma, \mathsf{ret}\ v \rrbracket_{id} \downarrow_\kappa \in \Sigma_1 \downarrow_\kappa$

By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

(3) $ExtCoreRs_1 \downarrow_\kappa \leq_c ExtCoreRs_2 \downarrow_\kappa$

(4) $\Psi_1 \downarrow_\kappa \leq_c \Psi_2 \downarrow_\kappa$

By (3)

(5) $ExtCoreRs_1 \downarrow_\kappa \leq ExtCoreRs_2 \downarrow_\kappa$

(6) $ExtCoreRs_2 = ExtCoreRs'_2 :: ExtCoreR_2$, s.t. $ExtCoreR_2 \downarrow_\kappa \leq ExtCoreR_1 \downarrow_\kappa$

(7) and $ExtCoreR_2 = ExtCoreR_{\mathit{ctx}}{}' \llbracket \Gamma, \mathsf{ret}\ v \rrbracket_{id} = (id, (\llbracket \Gamma \rrbracket, cmd, EventHandlers_2 :: EventHandler_{2ctx} \llbracket \mathsf{ret}\ v \rrbracket), \ell)$
and $ExtCoreR_{\mathit{ctx}} \llbracket \Gamma, \mathsf{ret}\ v \rrbracket_{id} \downarrow_\kappa \leq ExtCoreR'_{\mathit{ctx}} \llbracket \Gamma, \mathsf{ret}\ v \rrbracket_{id} \downarrow_\kappa$
and $EventHandlers_1 \downarrow_\kappa \leq EventHandlers_2 \downarrow_\kappa$
and $EventHandler_{1\mathit{ctx}} \llbracket \mathsf{ret}\ v \rrbracket \downarrow_\kappa \leq EventHandler_{2\mathit{ctx}} \llbracket \mathsf{ret}\ v \rrbracket \downarrow_\kappa$

Let $EventHandler_{2ctx} \llbracket \mathsf{ret}\ v \rrbracket = (id_{eh}, eventType, x.cmd, \mathcal{E}', \llbracket \mathsf{ret}\ v \rrbracket, id_e, \mathsf{blocking}, \mathsf{some}(e, id_r))$

By (7)

(8) $\mathcal{E} \downarrow_\kappa \leq \mathcal{E}' \downarrow_\kappa$

By (4), Lemma 1

(9) $\Psi_1 \downarrow_\kappa = \psi_1 :: \Psi_{11} :: \Psi_{12} :: \Psi_{13}$, where $\psi_1 \sim_b e$
Ψ_{11} contains states of the form $\mathsf{ProcBlkEvtState}(\_, \_, e)$
Ψ_{12} contains states of the form $\mathsf{DoneBlkEvtState}(\_, id, e, c)$
$\nexists \psi \in \Psi_{13}$, s.t. $\psi = \mathsf{ProcBlkEvtState}(\_, \_, e)$, or $\psi = \mathsf{DoneBlkEvtState}(\_, \_, e, \_)$

(10) $\forall \psi \in \psi_1 :: \Psi_{11} :: \Psi_{12}, labOf(\psi) = labOf(e)$

(11) $\psi_{1a} \in \psi_{11}$,

(12) $\Psi_2 \downarrow_\kappa = \psi_1 :: \Psi_{11} :: \Psi_{12} :: \Psi_{23}$
$\nexists \psi \in \Psi_{23}$, s.t. $\psi = \mathsf{ProcBlkEvtState}(\_, \_, e)$, or $\psi = \mathsf{DoneBlkEvtState}(\_, \_, e, \_)$

let $\Sigma_2 = \Sigma_{ctx2} \llbracket \Gamma, \mathsf{ret}\ v \rrbracket_{id}$, by (6), (11), (12), we can apply RETURN-CORE-BLOCKING to Σ_2,

(13) $\psi_{2b} = \mathsf{DoneBlkEvtState}(\ell^-, id_r, e, v) = \psi_{1b}$

(14) $\Sigma'_{ctx2} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id} = \Sigma_{ctx2} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id}[\Psi_2 \Leftarrow \Psi_2 :: \psi_{2b}]$

(15) $\Sigma_{ctx2} \llbracket \Gamma, \mathsf{ret}\ v \rrbracket_{id}; \mathcal{E}_2 \xrightarrow{\tau} \Sigma'_{ctx2} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id}; \mathcal{E}'_2, \mathcal{E}'_2 = \mathcal{E}_2$

RETURN-CORE-BLOCKING does not change the event handler other than the current command

(16) $ExtCoreR_{\mathit{ctx}} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id} \downarrow_\kappa \leq_c ExtCoreR_{\mathit{ctx}}' \llbracket \Gamma, \mathsf{skip} \rrbracket_{id} \downarrow_\kappa$

By (9), (13), BROWSERSTATE1, Lemma 10

(17) $(\Psi_1 :: \psi_{1b}) \downarrow_\kappa \leq_c (\Psi_2 :: \psi_{2b}) \downarrow_\kappa$

By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$, $\Sigma_1 = \Sigma_{ctx} \llbracket \Gamma, \mathsf{ret}\ v \rrbracket_{id}$, $\Sigma_2 = \Sigma_{ctx2} \llbracket \Gamma, \mathsf{ret}\ v \rrbracket_{id}$, (A6) (14), (17) and STATE

(18) $\Sigma'_{ctx} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id} \downarrow_\kappa \leq \Sigma'_{ctx2} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id} \downarrow_\kappa$

By assumption, (A7), (15)

(19) $\mathcal{E}'_1 \downarrow_\kappa \leq \mathcal{E}'_2 \downarrow_\kappa$

sub-subcase ii. $\ell^- \not\sqsubseteq \kappa$

By EXTCORE2

(1) $ExtCoreR \downarrow_\kappa = \cdot$, $ExtCoreR_{\mathit{ctx}} \llbracket \Gamma, \mathsf{ret}\ v \rrbracket_{id} \notin \Sigma_1 \downarrow_\kappa$

By BROWSERSTATE2, Lemma 1

(2) $\psi_{1b} \downarrow_\kappa = \cdot$, $ExtCoreR_{\mathit{ctx}} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id} \notin \Sigma'_1 \downarrow_\kappa$

(3) $\Psi'_1 \downarrow_\kappa = \Psi_1 \downarrow_\kappa \leq_c \Psi_2 \downarrow_\kappa$

By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$, $\Sigma_1 = \Sigma_{ctx} \llbracket \Gamma, \mathsf{ret}\ v \rrbracket_{id}$, and $\Sigma'_1 = \Sigma'_{ctx} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id}$, (2) and (3)

(4) $\Sigma'_1 \downarrow_\kappa = \Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

By assumption

(5) $\mathcal{E}'_1 \downarrow_\kappa = \mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$

case: SKIPAPICALL-S1

There are three subcases: (1) the script is a page script handler, (2) the script is a content script and (3) the script is in the extension core.

$$\frac{\begin{array}{l}\ell = \mathsf{ctxOfId}(\Sigma_{ctx} \llbracket \Gamma, API_s(\vec{v}) \rrbracket_{id}, id) \\ \ell^- \not\sqsubseteq \mathsf{labOf}(API_s) \qquad \text{the last argument of } \vec{v} \text{ is not a label}\end{array}}{\Sigma_{ctx} \llbracket \Gamma, API_s(\vec{v}) \rrbracket_{id}; \mathcal{E} \xrightarrow{\tau} \Sigma_{ctx} \llbracket \Gamma, \mathsf{void} \rrbracket_{id}; \mathcal{E}} \text{SKIPAPICALL-S1}$$

Subcase: the script is a page script handler.

Let $\Sigma_1 = (\Psi_1, Tabs_1 :: Tab_1, ExtCoreRs_1, progInjCSs_1, Exts_1, Cookies_1, MBookmarks_1, histories_1, UI_1)$
$\Sigma_2 = (\Psi_2, Tabs_2, ExtCoreRs_2, progInjCSs_2, Exts_2, Cookies_2, MBookmarks_2, histories_2, UI_2)$

By assumptions

$\Sigma_1 = \Sigma_{ctx} \llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id}$

$Tab_1 = Tab_{\mathit{ctx}} \llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id} =$

$(TabID_1, Docs_1 :: Doc_{env} \llbracket \Gamma \rrbracket, url_1, EventHandlers_1 :: EventHandler_{\mathit{ctx}} \llbracket API_s(\vec{e}) \rrbracket_{id}, \ell_1)$

and $\Sigma'_1 = \Sigma_{ctx} \llbracket \Gamma, \mathsf{void} \rrbracket_{id}$, $\mathcal{E}'_1 = \mathcal{E}_1$

Assume that the label of the node with ID id is ℓ_n: $\mathsf{ctxOfId}(\Sigma_1, id) = \ell_n$

sub-subcase i. $\ell_n{}^- \sqsubseteq \kappa$

By Lemma 5,

$id \in Docs_1 \downarrow_\kappa$, and exists $node_{env} \llbracket \Gamma \rrbracket_{id} \in Doc_{env} \llbracket \Gamma \rrbracket \downarrow_\kappa$

By definition of Prune,

(1) $EventHandler_{\mathit{ctx}} \llbracket API_s(\vec{e}) \rrbracket_{id} \in Tab_1 \downarrow_\kappa$

By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

(2) $Tabs_2 = Tabs'_2 :: Tab_2$ and $EventHandler_{\mathit{ctx}} \llbracket API_s(\vec{e}) \rrbracket_{id} \in Tab_2 \downarrow_\kappa$

By definitions and (2)

(3) exists $Docs'_2 \in Tab_2 \downarrow_\kappa$ and exists $node'_{env}\llbracket \Gamma \rrbracket_{id} \in Docs'_2$ s.t. $node_{\textbf{\textit{env}}}\llbracket \Gamma \rrbracket_{id} \leq_n node'_{env}\llbracket \Gamma \rrbracket_{id}$

By (2) and (3)

(4) $Tab_2 = (TabID_2, Docs_2\llbracket\Gamma\rrbracket, url_2, EventHandlers_2 :: EventHandler_{\textbf{\textit{ctx}}}\llbracket API_s(\vec{e}) \rrbracket_{id}, \ell_2)$

By (3), assumption, and $node'_{env}\llbracket \Gamma \rrbracket_{id} = (id, attributes', nodes', \llbracket\Gamma\rrbracket, \ell_n)$

(5) $\ell_n{}^- \not\sqsubseteq \mathsf{labOf}(API_s)$

Apply SKIPAPICALL-S1 to Σ_2, let $\Sigma_2 = \Sigma_{ctx2}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id}$

(6) $\Sigma_{ctx2}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id}; \mathcal{E}_2 \xrightarrow{\tau} \Sigma_{ctx2}\llbracket \Gamma, \mathsf{void} \rrbracket_{id}; \mathcal{E}_2$

By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$, $\Sigma_1 = \Sigma_{ctx}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id}$, $\Sigma_2 = \Sigma_{ctx2}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id}$,
and SKIPAPICALL-S1 does not change Σ_{ctx} and Σ_{ctx2}

(7) $\Sigma_{ctx}\llbracket \Gamma, \mathsf{void} \rrbracket_{id} \downarrow_\kappa \leq \Sigma_{ctx2}\llbracket \Gamma, \mathsf{void} \rrbracket_{id} \downarrow_\kappa$

By assumption

(8) $\mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$

(9) $\mathcal{E}'_1 \downarrow_\kappa = \mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$

sub-subcase ii. $\ell_n{}^- \not\sqsubseteq \kappa$

By definitions

(1) $id \notin Docs_1 \downarrow_\kappa$ and $EventHandler_{\textbf{\textit{ctx}}}\llbracket API_s(\vec{e}) \rrbracket_{id} \notin Tab_1 \downarrow_\kappa$ and $node_{\textbf{\textit{env}}}\llbracket \Gamma \rrbracket_{id} \notin Tab_1 \downarrow_\kappa$

By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$, $\Sigma_1 = \Sigma_{ctx}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id}$, and SKIPAPICALL-S1 does not change Σ_{ctx}

(2) $\Sigma_{ctx}\llbracket \Gamma, \mathsf{void} \rrbracket_{id} \downarrow_\kappa = \Sigma_{ctx}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id} \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

By assumption

(3) $\mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$

Subcase: the script is a content script handler

Let $\Sigma_1 = (\Psi_1, Tabs_1 :: Tab_1, ExtCoreRs_1, progInjCSs_1, Exts_1, Cookies_1, MBookmarks_1, histories_1, UI_1)$
$\Sigma_2 = (\Psi_2, Tabs_2, ExtCoreRs_2, progInjCSs_2, Exts_2, Cookies_2, MBookmarks_2, histories_2, UI_2)$

By assumptions

$\Sigma_1 = \Sigma_{ctx}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id}$

$Tab_1 = Tab_{\textbf{\textit{ctx}}}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id} = (TabID_1, Docs_1 :: Doc_{\textbf{\textit{ctx}}}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id}, url_1, EventHandlers_1, \ell_1)$

$Doc_{\textbf{\textit{ctx}}}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id} = (id_d, url, nodes, DocCSs :: DocCS_{\textbf{\textit{ctx}}}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id}, \ell_d)$

$DocCSs_{\textbf{\textit{ctx}}}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id} = (id_{ext}, id, id_t, \llbracket\Gamma\rrbracket, \llbracket API_s(\vec{e})\rrbracket, EventHandlers, \ell)$
or $(id_{ext}, id, id_t, \llbracket\Gamma\rrbracket, cmd, EventHandlers :: EventHandler_{\textbf{\textit{ctx}}}\llbracket API_s(\vec{e})\rrbracket, \ell)$

sub-subcase i. $\ell^- \sqsubseteq \kappa$

By DS1,

$DocCS_{\textbf{\textit{ctx}}}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id} \downarrow_\kappa = (id_{ext}, id, id_t, \llbracket\Gamma\rrbracket, \llbracket API_s(\vec{e})\rrbracket, EventHandlers \downarrow_\kappa, \ell)$
or $DocCS_{\textbf{\textit{ctx}}}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id} \downarrow_\kappa = (id_{ext}, id, id_t, \llbracket\Gamma\rrbracket, cmd, EventHandlers \downarrow_\kappa :: EventHandler_{\textbf{\textit{ctx}}}\llbracket API_s(\vec{e})\rrbracket \downarrow_\kappa, \ell)$

By definition

(1) $DocCS_{\textbf{\textit{ctx}}}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id} \downarrow_\kappa \in Tab_1 \downarrow_\kappa$

By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

(2) $Tabs_2 = Tabs'_2 :: Tab_2$ and $DocCS_{\textbf{\textit{ctx}}}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id} \downarrow_\kappa \in Tab_2 \downarrow_\kappa$

By definitions and (2)

(3) exists $DocCS'_{\textbf{\textit{ctx}}}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id} \in DocCSs'$, $DocCSs' \in Doc'$, $Doc' \in Docs'$, $Docs'_2 \in Tab_2 \downarrow_\kappa$
$DocCS'_{\textbf{\textit{ctx}}}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id} = (id_{ext}, id, id_t, \llbracket\Gamma\rrbracket, \llbracket API_s(\vec{e})\rrbracket, EventHandlers', \ell)$
or $(id_{ext}, id, id_t, \llbracket\Gamma\rrbracket, cmd, EventHandlers' :: EventHandler_{\textbf{\textit{ctx}}}\llbracket API_s(\vec{e})\rrbracket, \ell)$
s.t. $DocCS_{\textbf{\textit{ctx}}}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id} \leq DocCS'_{\textbf{\textit{ctx}}}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id}$

By (2) and (3)

(4) $Tab_2 = (TabID_2, Docs'_{\textbf{\textit{ctx}}}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id}, url_2, EventHandlers_2, \ell_2)$

By assumption

(5) $\ell^- \not\sqsubseteq \mathsf{labOf}(API_s)$

Apply SKIPAPICALL-S1 to Σ_2, let $\Sigma_2 = \Sigma_{ctx2}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id}$

(6) $\Sigma_{ctx2}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id}; \mathcal{E}_2 \xrightarrow{\tau} \Sigma_{ctx2}\llbracket \Gamma, \mathsf{void} \rrbracket_{id}; \mathcal{E}_2$

By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$, $\Sigma_1 = \Sigma_{ctx}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id}$, $\Sigma_2 = \Sigma_{ctx2}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id}$,
and SKIPAPICALL-S1 does not change Σ_{ctx} and Σ_{ctx2}

(7) $\Sigma_{ctx}\llbracket \Gamma, \mathsf{void} \rrbracket_{id} \downarrow_\kappa \leq \Sigma_{ctx2}\llbracket \Gamma, void \rrbracket_{id} \downarrow_\kappa$

By assumption

(8) $\mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$

(9) $\mathcal{E}_1' \downarrow_\kappa = \mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$

sub-subcase ii. $\ell^- \not\sqsubseteq \kappa$

By DS2

(1) $DocCS_{\mathsf{ctx}}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id} \notin Tab_1 \downarrow_\kappa$

By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$, $\Sigma_1 = \Sigma_{ctx}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id}$, and SKIPAPICALL-S1 does not change Σ_{ctx}

(2) $\Sigma_{ctx}\llbracket \Gamma, \mathsf{void} \rrbracket_{id} \downarrow_\kappa = \Sigma_{ctx}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id} \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

By assumption

(3) $\mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$

Subcase: the script is in extension core

Let $\Sigma_1 = (\Psi_1, Tabs_1, ExtCoreRs_1 :: ExtCoreR, progInjCSs_1, Exts_1, sMode_1)$
$\Sigma_2 = (\Psi_2, Tabs_2, ExtCoreRs_2, progInjCSs_2, Exts_2, Cookies_2, MBookmarks_2, histories_2, UI_2)$

By assumptions

$\Sigma_1 = \Sigma_{ctx}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id}$

$ExtCoreR = ExtCoreR_{\mathsf{ctx}}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id} = (id, (\llbracket\Gamma\rrbracket, \llbracket API_s(\vec{e})\rrbracket, EventHandlers), \ell)$
or $(id, (\llbracket\Gamma\rrbracket, cmd, EventHandlers :: EventHandler_{ctx}\llbracket API_s(\vec{e})\rrbracket), \ell)$

sub-subcase i. $\ell^- \sqsubseteq \kappa$

By EXTCORE1,

$ExtCoreR_{\mathsf{ctx}}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id} \downarrow_\kappa = (id, (\llbracket\Gamma\rrbracket, \llbracket API_s(\vec{e})\rrbracket, EventHandlers \downarrow_\kappa), \ell)$
or $ExtCoreR_{\mathsf{ctx}}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id} \downarrow_\kappa = (id, (\llbracket\Gamma\rrbracket, API_s(\vec{e}), EventHandlers \downarrow_\kappa :: EventHandler_{\mathsf{ctx}}\llbracket cmd\rrbracket \downarrow_\kappa), \ell)$

By definition

(1) $ExtCoreR_{\mathsf{ctx}}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id} \downarrow_\kappa \in \Sigma_1 \downarrow_\kappa$

By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

(2) $ExtCoreR_{\mathsf{ctx}}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id} \downarrow_\kappa \in ExtCoreRs_2 \downarrow_\kappa$

By definitions and (2)

(3) exists $ExtCoreR'_{ctx2}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id} \downarrow_\kappa \in ExtCoreRs_2 \downarrow_\kappa$
s.t. $ExtCoreR_{\mathsf{ctx}}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id} \downarrow_\kappa \leq ExtCoreR'_{ctx2}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id} \downarrow_\kappa$

By definitions and assumption

(4) $\mathsf{ctxOfId}(\Sigma_2, id) = \ell'$, $\ell' = \ell$, $\ell'^- \not\sqsubseteq \mathsf{labOf}(API_s)$

Apply SKIPAPICALL-S1 to Σ_2, let $\Sigma_2 = \Sigma_{ctx2}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id}$

(5) $\Sigma_{ctx2}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id}; \mathcal{E}_2 \xrightarrow{\tau} \Sigma_{ctx2}\llbracket \Gamma, \mathsf{void} \rrbracket_{id}; \mathcal{E}_2$

By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$, $\Sigma_1 = \Sigma_{ctx}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id}$, $\Sigma_2 = \Sigma_{ctx2}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id}$,
and SKIPAPICALL-S1 does not change Σ_{ctx} and Σ_{ctx2}

(6) $\Sigma_{ctx}\llbracket \Gamma, \mathsf{void} \rrbracket_{id} \downarrow_\kappa \leq \Sigma_{ctx2}\llbracket \Gamma, \mathsf{void} \rrbracket_{id} \downarrow_\kappa$

By assumption

(7) $\mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$

(8) $\mathcal{E}_1' \downarrow_\kappa = \mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$

sub-subcase ii. $\ell^- \not\sqsubseteq \kappa$

By EXTCORE2

(1) $ExtCoreR \downarrow_\kappa = \cdot$, $ExtCoreR_{\mathsf{ctx}}\llbracket \Gamma, API_s(\vec{e}) \rrbracket_{id} \notin \Sigma_1 \downarrow_\kappa$

By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$ $\Sigma_1 = \Sigma_{ctx} \| \Gamma, API_s(\vec{e}) \|_{id}$, and SKIPAPICALL-S1 does not change Σ_{ctx}

(2) $\Sigma_{ctx} \| \Gamma, \mathsf{void} \|_{id} \downarrow_\kappa = \Sigma_{ctx} \| \Gamma, API_s(\vec{e}) \|_{id} \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

By assumption

(3) $\mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$

case: SKIPAPICALL-S2, SKIPAPICALL-A1, SKIPAPICALL-A2, the proofs are similar to the proofs for SKIPAPICALL-S1.

case: FIREAPICALL-S1

$$\frac{\ell = \mathsf{ctxOfId}(\Sigma_{ctx} \| \Gamma, API_s(\vec{v}) \|_{id}, id) \qquad \text{the last argument of } \vec{v} \text{ is not a label} \qquad \ell^- \sqsubseteq \mathsf{labOf}(API_s) \qquad \Sigma_{ctx} \| \Gamma, API_s(\vec{v}) \|_{id} \xrightarrow{\beta}_{API_s} \Sigma'; \mathcal{E}'}{\Sigma_{ctx} \| \Gamma, API_s(\vec{v}) \|_{id}; \mathcal{E} \xrightarrow{\beta} \Sigma'; \mathcal{E} :: \mathcal{E}'} \text{FIREAPICALL-S1}$$

By assumptions

$\Sigma_1 = \Sigma_{ctx} \| \Gamma, API_s(\vec{e}) \|_{id}$

$\Sigma_{ctx} \| \Gamma, API_s(\vec{e}) \|_{id} \xrightarrow{\beta}_{API_s} \Sigma_1'; \mathcal{E}_1'$

$\Sigma_{ctx} \| \Gamma, API_s(\vec{e}) \|_{id}; \mathcal{E}_1 \xrightarrow{\beta} \Sigma_1'; \mathcal{E}_1 :: \mathcal{E}_1'$

By Lemma 15, $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

if $\beta = \tau$, then

either $\Sigma_2 \xrightarrow{\tau}_{API_s} \Sigma_2'; \mathcal{E}_2'$, $\Sigma_1' \downarrow_\kappa \leq \Sigma_2' \downarrow_\kappa$ and $\mathcal{E}_1' \downarrow_\kappa \leq \mathcal{E}_2' \downarrow_\kappa$

or $\Sigma_1' \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$ and $\mathcal{E}_1' \downarrow_\kappa = \cdot$

if $\beta = \alpha$, $\alpha \downarrow_\kappa = \cdot$, then $\Sigma_1' \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$ and $\mathcal{E}_1' \downarrow_\kappa = \cdot$

if $\beta = \alpha$, $\alpha \downarrow_\kappa = \alpha$

$\Sigma_2 \xrightarrow{\alpha}_{API_s} \Sigma_2'; \mathcal{E}_2'$,

$\Sigma_2; \mathcal{E}_2 \xrightarrow{\alpha} \Sigma_2'; \mathcal{E}_2 :: \mathcal{E}_2'$,

where $\Sigma_1' \downarrow_\kappa \leq \Sigma_2' \downarrow_\kappa$ and $\mathcal{E}_1' \downarrow_\kappa \leq \mathcal{E}_2' \downarrow_\kappa$.

case: FIREAPICALL-S2.The proofs are similar to the proofs for FIREAPICALL-S1.

case: FIREAPICALL-A-CORE

$$\frac{\begin{array}{l}\Sigma_{ctx}(id) = (\Psi, \mathit{Tabs}, \mathit{ExtCoreRs}_{\mathbf{ctx}}(id), \cdots) \qquad \mathit{ExtCoreR}_{\mathbf{ctx}}(id) \in \mathit{ExtCoreRs}_{\mathbf{ctx}}(id) \\ \mathit{ExtCoreR}_{\mathbf{ctx}} \| \Gamma, API_a(\vec{v}) \|_{id} = (id, \_, \mathit{EventHandlers}, \ell) \\ \mathsf{labOf}(API_a) \sqsubseteq \ell^* \qquad \text{when } \vec{v} = \cdots, \kappa_t, \ell \Rightarrow \kappa_t \qquad \mathit{callbackOf}(\vec{v}) = x.cmd \\ \mathit{fresh}(id_{cb}) \qquad \mathit{fresh}(id_h) \qquad eh = (id_h, id_{cb}, x.cmd, \cdot, \mathsf{skip}, \mathsf{void}, \mathsf{nonblocking}, \mathsf{none}) \\ \mathit{ExtCoreR}_{\mathbf{ctx}}{}'(id) = \mathit{ExtCoreR}_{\mathbf{ctx}}(id)[\mathit{EventHandlers} \Leftarrow \mathit{EventHandlers} :: eh] \\ \mathit{ExtCoreRs}_{\mathbf{ctx}}{}'(id) = \mathit{ExtCoreRs}_{\mathbf{ctx}}(id)[\mathit{ExtCoreRs}_{\mathbf{ctx}}(id) \Leftarrow \mathit{ExtCoreRs}_{\mathbf{ctx}}{}'(id)] \\ \psi = \mathit{ayncAPIstate}(\ell^-, id_{cb}, API_a(\vec{v})) \\ \Sigma_{ctx}'(id) = \Sigma_{ctx}(id)[\mathit{ExtCoreRs}_{\mathbf{ctx}}(id) \Leftarrow \mathit{ExtCoreRs}_{\mathbf{ctx}}'(id)][\Psi \Leftarrow \Psi :: \psi]\end{array}}{\Sigma_{ctx} \| \Gamma, API_a(\vec{v}) \|_{id}; \mathcal{E} \xrightarrow{\tau} \Sigma_{ctx}' \| \Gamma, \mathsf{skip} \|_{id}; \mathcal{E}} \text{FIREAPICALL-A-CORE}$$

Let $\Sigma_1 = (\Psi_1, \mathit{Tabs}_1, \mathit{ExtCoreRs}_1 :: \mathit{ExtCoreR}, \mathit{progInjCSs}_1, \mathit{Exts}_1, \mathit{Cookies}_1, \mathit{MBookmarks}_1, \mathit{histories}_1, \mathit{UI}_1)$

$\Sigma_2 = (\Psi_2, \mathit{Tabs}_2, \mathit{ExtCoreRs}_2, \mathit{progInjCSs}_2, \mathit{Exts}_2, \mathit{Cookies}_2, \mathit{MBookmarks}_2, \mathit{histories}_2, \mathit{UI}_2)$

By assumptions

$\Sigma_1 = \Sigma_{ctx} \| \Gamma, API_a(\vec{v}) \|_{id}$

$\mathit{ExtCoreR} = \mathit{ExtCoreR}_{\mathbf{ctx}} \llbracket \Gamma, \mathit{API}_a(\vec{v}) \rrbracket_{id} = (id, (\llbracket\Gamma\rrbracket, \llbracket \mathit{API}_a(\vec{v}) \rrbracket, \mathit{EventHandlers}), \ell)$
or $(id, (\llbracket\Gamma\rrbracket, cmd, \mathit{EventHandlers} :: \mathit{EventHandler}_{ctx}\llbracket \mathit{API}_a(\vec{v})\rrbracket), \ell)$
$\mathit{callbackOf}(\vec{v}) = x.cmd, \mathit{fresh}(id_{cb})$
$eh = (id_{cb}, x.cmd, \cdot, \mathsf{skip}, \mathsf{void}, \mathsf{nonblocking}, \mathsf{none})$
$\mathit{ExtCoreR}_{\mathbf{ctx}}{}'(id) = \mathit{ExtCoreR}_{\mathbf{ctx}}(id)[\mathit{EventHandlers} \Leftarrow \mathit{EventHandlers} :: eh]$
$\psi = \mathit{ayncAPIstate}(\ell^-, id_{cb}, \mathit{API}_a(\vec{v}))$
$\Sigma'_{ctx}(id) = \Sigma_{ctx}(id)[\mathit{ExtCoreR}_{\mathbf{ctx}}(id) \Leftarrow \mathit{ExtCoreR}'_{\mathbf{ctx}}(id)][\Psi \Leftarrow \Psi :: \psi]$
$\Sigma'_1 = \Sigma'_{ctx}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}$

sub-subcase i. $\ell^- \sqsubseteq \kappa$

By ExtCore1,
$\mathit{ExtCoreR}_{\mathbf{ctx}} \llbracket \Gamma, \mathit{API}_a(\vec{v}) \rrbracket_{id} \downarrow_\kappa = (id, (\llbracket\Gamma\rrbracket, \llbracket \mathit{API}_a(\vec{v}) \rrbracket, \mathit{EventHandlers}\downarrow_\kappa), \ell)$
or $\mathit{ExtCoreR}_{\mathbf{ctx}} \llbracket \Gamma, \mathit{API}_a(\vec{v}) \rrbracket_{id} \downarrow_\kappa = (id, (\llbracket\Gamma\rrbracket, \mathit{API}_a(\vec{v}), \mathit{EventHandlers}\downarrow_\kappa :: \mathit{EventHandler}_{\mathbf{ctx}}\llbracket cmd \rrbracket\downarrow_\kappa), \ell)$

By definition
(1) $\mathit{ExtCoreR}_{\mathbf{ctx}} \llbracket \Gamma, \mathit{API}_a(\vec{v}) \rrbracket_{id} \downarrow_\kappa \in \Sigma_1 \downarrow_\kappa$

By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$
(2) $\mathit{ExtCoreR}_{\mathbf{ctx}} \llbracket \Gamma, \mathit{API}_a(\vec{v}) \rrbracket_{id} \downarrow_\kappa \in \mathit{ExtCoreRs}_2 \downarrow_\kappa$

By definitions and (2)
(3) exists $\mathit{ExtCoreR}_{ctx2} \llbracket \Gamma, \mathit{API}_a(\vec{v}) \rrbracket_{id} \downarrow_\kappa \in \mathit{ExtCoreRs}_2 \downarrow_\kappa$
s.t. $\mathit{ExtCoreR}_{\mathbf{ctx}} \llbracket \Gamma, \mathit{API}_a(\vec{v}) \rrbracket_{id} \downarrow_\kappa \leq \mathit{ExtCoreR}_{ctx2} \llbracket \Gamma, \mathit{API}_a(\vec{v}) \rrbracket_{id} \downarrow_\kappa$
(4) $\mathit{ExtCore}_{ctx2} \llbracket \Gamma, \mathit{API}_a(\vec{v}) \rrbracket_{id} = (id, (\llbracket\Gamma\rrbracket, \llbracket \mathit{API}_a(\vec{v}) \rrbracket, \mathit{EventHandlers}_2), \ell)$
or $(id, (\llbracket\Gamma\rrbracket, cmd, \mathit{EventHandlers}_2 :: \mathit{EventHandler}_{ctx2}\llbracket \mathit{API}_a(\vec{v})\rrbracket), \ell)$

Apply FireAPICall-A-Core to Σ_2, let $\Sigma_2 = \Sigma_{ctx2}\llbracket \Gamma, \mathit{API}_a(\vec{v}) \rrbracket_{id}$
(5) $\Sigma_{ctx2}\llbracket \Gamma, \mathit{API}_a(\vec{v}) \rrbracket_{id}; \mathcal{E}_2 \xrightarrow{\tau} \Sigma'_{ctx2}\llbracket \Gamma, \mathsf{void} \rrbracket_{id}; \mathcal{E}_2$
(6) Let $id_{cb2} = id_{cb}$, then $eh_2 = (id_{cb2}, x.cmd, \cdot, \mathsf{skip}, \mathsf{void}, \mathsf{nonblocking}, \mathsf{none}) = eh$
(7) $\mathit{ExtCoreR}'_{ctx2}(id) = \mathit{ExtCoreR}_{ctx2}(id)[\mathit{EventHandlers}_2 \Leftarrow \mathit{EventHandlers}_2 :: eh_2]$
(8) $\mathit{ExtCoreRs}'_{ctx2}(id) = \mathit{ExtCoreRs}_{ctx2}(id)[\mathit{ExtCoreR}_{ctx2}(id) \Leftarrow \mathit{ExtCoreR}'_{ctx2}(id)]$
(9) $\psi_2 = \psi = \mathit{ayncAPIstate}(\ell^-, id_{cb}, \mathit{API}_a(\vec{v}))$
(10) $\Sigma'_{ctx2}(id) = \Sigma_{ctx2}(id)[\mathit{ExtCoreRs}_{ctx2}(id) \Leftarrow \mathit{ExtCoreRs}'_{ctx2}(id)][\Psi_2 \Leftarrow \Psi_2 :: \psi_2]$

By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$, $\Sigma_1 = \Sigma_{ctx}\llbracket \Gamma, \mathit{API}_a(\vec{v}) \rrbracket_{id}$, $\Sigma_2 = \Sigma_{ctx2}\llbracket \Gamma, \mathit{API}_a(\vec{v}) \rrbracket_{id}$, (3), (6), (7), (8), (9), (10),
ExtCore1, BrowserState1, EH, and the new states are not related to blocking states
(11) $\Sigma'_{ctx}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} \downarrow_\kappa \leq \Sigma'_{ctx2}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} \downarrow_\kappa$

By assumption
(12) $\mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$

sub-subcase ii. $\ell^- \not\sqsubseteq \kappa$

By ExtCore2 and BrowserState2
(1) $\mathit{ExtCoreR} \downarrow_\kappa = \cdot$, $\mathit{ExtCoreR}_{\mathbf{ctx}} \llbracket \Gamma, \mathit{API}_s(\vec{v}) \rrbracket_{id} \notin \Sigma_1 \downarrow_\kappa$
(2) $\mathit{ExtCoreR}' \downarrow_\kappa = \cdot$, $\mathit{ExtCoreR}_{\mathbf{ctx}}{}' \llbracket \Gamma, \mathsf{skip} \rrbracket_{id} \notin \Sigma'_1 \downarrow_\kappa$
(3) $\psi \downarrow_\kappa = \cdot$

By assumptions, $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$ $\Sigma_1 = \Sigma_{ctx}\llbracket \Gamma, \mathit{API}_a(\vec{v}) \rrbracket_{id}$, $\Sigma'_1 = \Sigma'_{ctx}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}$, and
$\Sigma'_{ctx}(id) = \Sigma_{ctx}(id)[\mathit{ExtCoreR}_{\mathbf{ctx}}(id) \Leftarrow \mathit{ExtCoreR}'_{\mathbf{ctx}}(id)][\Psi \Leftarrow \Psi :: \psi]$
(4) $\Sigma'_{ctx}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} \downarrow_\kappa = \Sigma_{ctx}\llbracket \Gamma, \mathit{API}_a(\vec{v}) \rrbracket_{id} \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

By assumption
(5) $\mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$

case: FireAPICall-A-CS and FireAPICall-A-Page. The proofs are similar to the proofs for FireAPICall-A-Core.

case: EventFireNonBL

$$\frac{\begin{array}{l} e \text{ is a blocking event} \quad \Sigma = (\Psi, \mathit{Tabs}, \mathit{ExtCoreRs}, \cdots) \quad \Psi = \Psi' :: \psi \quad \psi \sim_b e \\ \forall \mathit{ExtCoreR} \in \mathit{ExtCoreRs}, \mathit{EHs} = \mathit{getActiveHandlers}(\mathit{ExtCoreR}, e), \\ \quad \forall eh \in \mathit{EHs}, \mathsf{labOf}(eh)^{-} = \mathsf{labOf}(e) \Rightarrow \mathit{isBlocking}(eh) = \mathsf{false} \\ (\psi', e') = \mathit{nextState}(\psi, \mathsf{void}) \quad \Sigma' = \Sigma[\Psi \Leftarrow \Psi' :: \psi'][\mathit{ExtCoreRs} \Leftarrow \mathit{ExtCoreRs} \lhd_Q e] \end{array}}{\Sigma; \mathcal{E} :: e \xrightarrow{\tau} \Sigma'; \mathcal{E} :: e'} \text{EVENTFIRENONBL}$$

By assumptions
(A1) $\Sigma_1 = (\Psi_1, \mathit{Tabs}_1, \mathit{ExtCoreRs}_1, \cdots)$, $\Psi_1 = \Psi_1'' :: \psi_1$
(A2) $\mathcal{E}_1 = \mathcal{E}_1'' :: e_1$, $\psi_1 \sim_b e_1$
(A3) $\forall \mathit{ExtCoreR} \in \mathit{ExtCoreRs}_1, \mathit{EHs} = \mathit{getActiveHandlers}(\mathit{ExtCoreR}, e_1)$,
$\quad \forall eh \in \mathit{EHs}, \mathsf{labOf}(eh)^{-} = \mathsf{labOf}(e) \Rightarrow \mathit{isBlocking}(eh) = \mathsf{false}$
(A4) $(\psi_1', e_1') = \mathit{nextState}(\psi_1, \mathsf{void})$
(A5) $\Sigma_1' = \Sigma_1[\Psi_1 \Leftarrow \Psi_1'' :: \psi_1'][\mathit{ExtCoreRs}_1 \Leftarrow \mathit{ExtCoreRs}_1 \lhd_Q e_1]$
(A6) $\mathcal{E}_1' = \mathcal{E}_1'' :: e_1'$
(A7) $\Sigma_2 = (\Psi_2, \mathit{Tabs}_2, \mathit{ExtCoreRs}_2, \cdots)$

sub-subcase i. $\mathit{labOf}(e_1) \sqsubseteq \kappa$
By $\psi_1 \sim_b e_1$, Lemma 1, BROWSERSTATE1
(1) $\mathit{labOf}(\psi_1) = \mathit{labOf}(e_1) \sqsubseteq \kappa$, $\psi_1 \downarrow_\kappa = \psi_1$
By $\mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$, $\mathit{labOf}(e_1) \sqsubseteq \kappa$, EVENT1
(2) $e_1 \downarrow_\kappa = e_1$, $\exists e_2 \in \mathcal{E}_2$, s.t. $\mathcal{E}_2 = \mathcal{E}_2'' :: e_2$, $\mathcal{E}_1'' \downarrow_\kappa \leq \mathcal{E}_2'' \downarrow_\kappa$, $e_1 \downarrow_\kappa \leq e_2 \downarrow_\kappa$, that is $e_2 = e_1$.
By $\Psi_1 \downarrow_\kappa \leq_c \Psi_2 \downarrow_\kappa$, e_1 is a blocking event, $e_2 = e_1$
(3) $\Psi_1 \downarrow_\kappa \leq_{c,e_1} \Psi_2 \downarrow_\kappa$
(4) $\Psi_1 \downarrow_\kappa \leq \Psi_2 \downarrow_\kappa$
(5) $\Psi_1 \downarrow_\kappa = \psi_1 :: \Psi_{11} :: \Psi_{12} :: \Psi_{13}$
Ψ_{11} contains states of the form $\mathsf{ProcBlkEvtState}(\_, \_, e_1)$
Ψ_{12} contains states of the form $\mathsf{DoneBlkEvtState}(\_, \_, e_1, \_)$
$\nexists \psi \in \Psi_{13}$, s.t. $\psi = \mathsf{ProcBlkEvtState}(\_, \_, e_1)$, or $\psi = \mathsf{DoneBlkEvtState}(\_, \_, e_1, \_)$
(6) $\Psi_2 \downarrow_\kappa = \psi_1 :: \Psi_{11} :: \Psi_{12} :: \Psi_{23}$
$\nexists \psi \in \Psi_{23}$, s.t. $\psi = \mathsf{ProcBlkEvtState}(\_, \_, e_2)$, or $\psi = \mathsf{DoneBlkEvtState}(\_, \_, e_2, \_)$
By EVENTFIRENONBL is an event firing rule, Lemma 9
(7) $\Psi_{11} = \cdot$, $\Psi_{12} = \cdot$
(8) $\Psi_{13} \leq_c \Psi_{23}$
By $\psi_1 \sim_b e_1$, $\psi_1 \in \Psi_2 \downarrow_\kappa$, $e_1 = e_2$,
(9) $\psi_1 \in \Psi_2$, and $\psi_1 \sim_b e_2$, $\Psi_2 = \Psi_2'' :: \psi_1$
By $\mathit{ExtCoreRs}_1 \downarrow_\kappa \leq_c \mathit{ExtCoreRs}_2 \downarrow_\kappa$,
(10) $\mathit{ExtCoreRs}_1 \downarrow_\kappa \leq \mathit{ExtCoreRs}_2 \downarrow_\kappa$
$\mathit{ExtCoreRs}_1 \downarrow_\kappa \leq_{c,e_1} \mathit{ExtCoreRs}_2 \downarrow_\kappa$,
$\mathit{ExtCoreRs}_1 \downarrow_\kappa = \mathit{ExtCoreRs}_{b1} :: \mathit{ExtCoreRs}_{nb1}$,
$\mathit{ExtCoreRs}_2 \downarrow_\kappa = \mathit{ExtCoreRs}_{b2} :: \mathit{ExtCoreRs}_{nb2}$
$\mathit{ExtCoreRs}_{b1} \leq_{B,e_1} \mathit{ExtCoreRs}_{b2}$
every core in $\mathit{ExtCoreRs}_{b1}$ and $\mathit{ExtCoreRs}_{b2}$ contains a blocking event handler for e_1,
no blocking event handlers for e_1 in $\mathit{ExtCoreRs}_{nb1}$ and $\mathit{ExtCoreRs}_{nb2}$
(11) $\forall \mathit{ExtCoreR} \in \mathit{ExtCoreRs}_1 \downarrow_\kappa, \exists \mathit{ExtCoreR}' \in \mathit{ExtCoreRs}_2 \downarrow_\kappa$, s.t. $\mathit{ExtCoreR} \downarrow_\kappa \leq \mathit{ExtCoreR}' \downarrow_\kappa$
By $\mathit{ExtCoreRs}_{b1} \leq_{B,e_1} \mathit{ExtCoreRs}_{b2}$
(12) There are matching pairs of $\mathit{ExtCoreR}$ and $\mathit{ExtCoreR}'$
s.t. $\mathit{ExtCoreR} \in \mathit{ExtCoreRs}_{b1}$, $\mathit{ExtCoreR}' \in \mathit{ExtCoreRs}_{b2}$
$\mathit{ExtCoreR} = (id_{ext}, (\_, \_, \mathit{EventHandlers}_1 :: eh_1), \ell)$

$ExtCoreR' = (id_{ext}, (\_, \_, EventHandlers_2 :: eh_2), \ell)$
$eh_1 = (eventTypeOf(e), \_, eventQueue_1, \_, \_, \mathsf{blocking}, \_)$
$eh_2 = (eventTypeOf(e), \_, eventQueue_2, \_, \_, \mathsf{blocking}, \_)$
$ExtCoreR \leq ExtCoreR'$

By (A3), (12), $e_1 = e_2$

(13) $\forall ExtCoreR' \in ExtCoreRs_2, EHs = getActiveHandlers(ExtCoreR', e_2),$
$\forall eh \in EHs, \mathsf{labOf}(eh)^{-} = \mathsf{labOf}(e_2) \Rightarrow isBlocking(eh) = \mathsf{false}$
This can be show by contradiction: if (11) is not true, then (A3) is not satisfied.

By (9), (13), apply EVENTFIRENONBL to Σ_2

(14) $(\psi_1', e_1') = nextState(\psi_1, \mathsf{void})$
(15) $\Sigma_2' = \Sigma_2[\Psi_2 \Leftarrow \Psi_2'' :: \psi_1'][ExtCoreRs_2 \Leftarrow ExtCoreRs_2 \lhd_Q e_2]$
(16) $\Sigma_2; \mathcal{E}_2 \xrightarrow{\tau} \Sigma_2'; \mathcal{E}_2', \mathcal{E}_2' = \mathcal{E}_2'' :: e_1'$

By $\Psi_{13} \leq_c \Psi_{23}$, Lemma 10

(17) $\Psi_{13} :: \psi_1' \leq_c \Psi_{23} :: \psi_1'$
$(\Psi_1'' :: \psi_1') \downarrow_\kappa = \Psi_{13} :: \psi_1' \leq_c (\Psi_2'' :: \psi_1') \downarrow_\kappa = \Psi_{23} :: \psi_1'$

By $ExtCoreRs_1 \downarrow_\kappa \leq ExtCoreRs_2 \downarrow_\kappa$, $e_1 = e_2$, Lemma 3

(18) $(ExtCoreRs_1 \lhd_Q e_1) \downarrow_\kappa \leq_c (ExtCoreRs_2 \lhd_Q e_2) \downarrow_\kappa$

By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$, (15), (17), (18), STATE

(19) $\Sigma_1' \downarrow_\kappa \leq \Sigma_2' \downarrow_\kappa$

By (A6), (2), (15)

(20) $\mathcal{E}_1' \downarrow_\kappa \leq \mathcal{E}_2' \downarrow_\kappa$

sub-subcase ii. $labOf(e_1) \not\sqsubseteq \kappa$

By Lemma 2

(1) $ExtCoreRs_1 \lhd_Q e_1 \downarrow_\kappa = ExtCoreRs_1 \downarrow_\kappa \leq_c ExtCoreRs_2 \downarrow_\kappa$

By (A4), Lemma 13

(2) $\psi_1 \downarrow_\kappa = \cdot, \psi_1' \downarrow_\kappa = \cdot, e_1' \downarrow_= \cdot$
(3) $\Psi_1' \downarrow_\kappa = \Psi_1 \downarrow_\kappa \leq_c \Psi_2 \downarrow_\kappa$ By (1), (3), (A5)
(4) $\Sigma_1' \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$
(4) $\mathcal{E}_1' \downarrow_\kappa = \mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$

case: EVENTFIREBL satisfies Lemma 16.

$$\frac{\begin{array}{l} e \text{ is a blocking event} \qquad \Sigma = (\Psi :: \psi, Tabs, ExtCoreRs, \cdots) \qquad \psi \sim_b e \\ \exists ExtCoreR \in ExtCoreRs, EHs = getActiveHandlers(ExtCoreR, e), \\ \quad \exists eh \in EHs, isBlocking(eh) = \mathsf{true} \wedge \mathsf{labOf}(eh)^{-} = \mathsf{labOf}(e) \\ \Sigma' = \Sigma[ExtCoreRs \Leftarrow ExtCoreRs \lhd_Q e] \end{array}}{\Sigma; \mathcal{E} :: e \xrightarrow{\tau} \Sigma'; \mathcal{E}} \text{ EVENTFIREBL}$$

By assumptions

(A1) $\Sigma_1 = (\Psi_1, Tabs_1, ExtCoreRs_1, \cdots), \Psi_1 = \Psi_1'' :: \psi_1$
(A2) $\mathcal{E}_1 = \mathcal{E}_1'' :: e_1, \psi_1 \sim_b e_1$
(A3) $\exists ExtCoreR \in ExtCoreRs_1, EHs = getActiveHandlers(ExtCoreR, e_1),$
$\exists eh \in EHs, isBlocking(eh) = \mathsf{true}$
(A4) $\Sigma_1' = \Sigma_1[ExtCoreRs_1 \Leftarrow ExtCoreRs_1 \lhd_Q e_1]$
(A5) $\mathcal{E}_1' = \mathcal{E}_1''$
(A6) $\Sigma_2 = (\Psi_2, Tabs_2, ExtCoreRs_2, \cdots)$

sub-subcase i. $labOf(e_1) \sqsubseteq \kappa$

By $\psi_1 \sim_b e_1$, Lemma 1, BROWSERSTATE1

(1) $\quad labOf(\psi_1) = labOf(e_1) \sqsubseteq \kappa$, $\psi_1 \downarrow_\kappa = \psi_1$

By $\mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$, $labOf(e_1) \sqsubseteq \kappa$, EVENT1

(2) $\quad e_1 \downarrow_\kappa = e_1, \exists e_2 \in \mathcal{E}_2$, s.t. $\mathcal{E}_2 = \mathcal{E}_2'' :: e_2$, $\mathcal{E}_1'' \downarrow_\kappa \leq \mathcal{E}_2'' \downarrow_\kappa$, $e_1 \downarrow_\kappa \leq e_2 \downarrow_\kappa$, that is $e_2 = e_1$.

By $\Psi_1 \downarrow_\kappa \leq_c \Psi_2 \downarrow_\kappa$, e_1 is a blocking event, $e_2 = e_1$

(3) $\quad \Psi_1 \downarrow_\kappa \leq_{c,e_1} \Psi_2 \downarrow_\kappa$

(4) $\quad \Psi_1 \downarrow_\kappa \leq \Psi_2 \downarrow_\kappa$

(5) $\quad \Psi_1 \downarrow_\kappa = \psi_1 :: \Psi_{11} :: \Psi_{12} :: \Psi_{13}$

Ψ_{11} contains states of the form $\mathsf{ProcBlkEvtState}(\_,\_,e_1)$

Ψ_{12} contains states of the form $\mathsf{DoneBlkEvtState}(\_,\_,e_1,\_)$

$\nexists \psi \in \Psi_{13}$, s.t. $\psi = \mathsf{ProcBlkEvtState}(\_,\_,e_1)$, or $\psi = \mathsf{DoneBlkEvtState}(\_,\_,e_1,\_)$

(6) $\quad \Psi_2 \downarrow_\kappa = \psi_1 :: \Psi_{11} :: \Psi_{12} :: \Psi_{23}$

$\nexists \psi \in \Psi_{23}$, s.t. $\psi = \mathsf{ProcBlkEvtState}(\_,\_,e_2)$, or $\psi = \mathsf{DoneBlkEvtState}(\_,\_,e_2,\_)$

By $\psi_1 \sim_b e_1$, $\psi_1 \in \Psi_2 \downarrow_\kappa$, $e_1 = e_2$,

(7) $\quad \psi_1 \in \Psi_2$, and $\psi_1 \sim_b e_2$

By $ExtCoreRs_1 \downarrow_\kappa \leq_c ExtCoreRs_2 \downarrow_\kappa$,

(8) $\quad ExtCoreRs_1 \downarrow_\kappa \leq ExtCoreRs_2 \downarrow_\kappa$

$ExtCoreRs_1 \downarrow_\kappa \leq_{c,e_1} ExtCoreRs_2 \downarrow_\kappa$,

$ExtCoreRs_1 \downarrow_\kappa = ExtCoreRs_{b1} :: ExtCoreRs_{nb1}$,

$ExtCoreRs_2 \downarrow_\kappa = ExtCoreRs_{b2} :: ExtCoreRs_{nb2}$

$ExtCoreRs_{b1} \leq_{B,e_1} ExtCoreRs_{b2}$

every core in $ExtCoreRs_{b1}$ and $ExtCoreRs_{b2}$ contains a blocking event handler for e_1,

no blocking event handlers for e_1 in $ExtCoreRs_{nb1}$ and $ExtCoreRs_{nb2}$

(9) $\quad \forall ExtCoreR \in ExtCoreRs_1 \downarrow_\kappa, \exists ExtCoreR' \in ExtCoreRs_2 \downarrow_\kappa$, s.t. $ExtCoreR \downarrow_\kappa \leq ExtCoreR' \downarrow_\kappa$

By $labOf(e_1) \sqsubseteq \kappa$, and $\forall eh \in ExtCoreR$, $labOf(eh) = labOf(ExtCoreR)$

(10) $\quad \forall ExtCore \in ExtCoreRs_1$, s.t. $EHs = getActiveHandlers(ExtCoreR, e_1)$,

$\exists eh \in EHs, isBlocking(eh) = \mathsf{true}$

$\wedge \mathsf{labOf}(eh)^- = \mathsf{labOf}(e_1)$

$labOf(extCoreR) \sqsubseteq \kappa$, $ExtCoreR \in ExtCoreRs_{b1}$

(11) $\quad ExtCoreRs_{b1} \neq \cdot$

By $ExtCoreRs_{b1} \leq_{B,e_1} ExtCoreRs_{b2}$

(12) $\quad \forall ExtCoreR \in ExtCoreRs_{b1}$, there is $ExtCoreR' \in ExtCoreRs_{b2}$

s.t. $\quad ExtCoreR = (id_{ext}, (\_,\_, EventHandlers_1 :: eh_1), \_)$

$ExtCoreR' = (id_{ext}, (\_,\_, EventHandlers_2 :: eh_2), \_)$

$eh_1 = (eventTypeOf(e), \_, eventQueue_1, \_,\_,\_, \mathsf{blocking}, \_)$

$eh_2 = (eventTypeOf(e), \_, eventQueue_2, \_,\_,\_, \mathsf{blocking}, \_)$

$ExtCoreR \leq ExtCoreR'$

By (2), (10), (11), (12), $e_1 = e_2$

(13) $\quad \exists ExtCoreR' \in ExtCoreRs_2, EHs = getActiveHandlers(ExtCoreR, e_2)$,

$\exists eh \in EHs, isBlocking(eh) = \mathsf{true} \wedge \mathsf{labOf}(eh)^- = \mathsf{labOf}(e_2)$

By (7), (13), apply EVENTFIREBL to Σ_2

(14) $\quad \Sigma_2; \mathcal{E}_2 \xrightarrow{\tau} \Sigma_2'; \mathcal{E}_2', \mathcal{E}_2' = \mathcal{E}_2''$

(15) $\quad \Sigma_2' = \Sigma_2[ExtCoreRs_2 \Leftarrow ExtCoreRs_2 \lhd_Q e_2]$

By $ExtCoreRs_1 \downarrow_\kappa \leq_c ExtCoreRs_2 \downarrow_\kappa$, $e_1 = e_2$, Lemma 3

(16) $\quad (ExtCoreRs_1 \lhd_Q e_1) \downarrow_\kappa \leq_c (ExtCoreRs_2 \lhd_Q e_2) \downarrow_\kappa$

By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$, (16), STATE

(17) $\quad \Sigma_1' \downarrow_\kappa \leq \Sigma_2' \downarrow_\kappa$

By (A5), (2), (14)

(18) $\quad \mathcal{E}_1' \downarrow_\kappa \leq \mathcal{E}_2' \downarrow_\kappa$

sub-subcase ii. $labOf(e_1) \not\sqsubseteq \kappa$

By Lemma 2

(1) $ExtCoreRs_1 \lhd_Q e_1 \downarrow_\kappa = ExtCoreRs_1 \downarrow_\kappa$

By (A4), (1)

(2) $\Sigma'_1 \downarrow_\kappa = \Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

By (A2), (A5)

(3) $\mathcal{E}'_1 \downarrow_\kappa \leq \mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$

case: BLOCKFINISHED1

$$\dfrac{\begin{array}{l} e \text{ is a blocking event} \qquad e \sim_b \psi \qquad \Sigma = (\Psi, Tabs, ExtCoreRs, \cdots) \qquad \Psi = \Psi_1 :: \psi :: \Psi_2 \\ \nexists \psi \in \Psi_1, s.t. \psi = \mathsf{ProcBlkEvtState}(\_, \_, e) \text{ or } \mathsf{DoneBlkEvtState}(\cdots, e, \_) \\ \Psi_2 \text{ contains matching pairs of } \mathsf{ProcBlkEvtState}(\_, id_r, e) \text{ and } \mathsf{DoneBlkEvtState}(\_, id_r, e, 0) \\ \forall ExtCoreR \in ExtCoreRs, EHs = getActiveHandlers(ExtCoreR, e), \\ \quad \forall eh \in EHs, labOf(eh)^- = labOf(e) \wedge isBlocking(eh) = \mathsf{true}, e \notin Qof(eh) \\ (\psi', \mathcal{E}_1) = \mathsf{nextState}(\psi, \mathsf{noChange}) \qquad \Sigma' = \Sigma[\Psi \Leftarrow \Psi_1 :: \psi'] \end{array}}{\Sigma; \mathcal{E} \xrightarrow{\tau} \Sigma'; \mathcal{E} :: \mathcal{E}_1} \text{ BLOCKFINISHED1}$$

By assumptions

(A1) $\Sigma_1 = (\Psi_1, Tabs_1, ExtCoreRs_1, \cdots)$

(A2) $\Psi_1 = \Psi_{11} :: \psi_1 :: \Psi_{12}$
$\psi_1 \sim_b e_1$, e_1 is a blocking event
$\nexists \psi \in \Psi_{11}, s.t. \psi = \mathsf{ProcBlkEvtState}(\_, \_, e_1)$ or $\mathsf{DoneBlkEvtState}(\cdots, e, \_)$
Ψ_{12} contains matching pairs of $\mathsf{ProcBlkEvtState}(\_, id_r, e_1)$ and $\mathsf{DoneBlkEvtState}(\_, id_r, e_1, 0)$
$labOf(\psi_1) = labOf(e_1)$

(A3) $\forall \psi \in \Psi_{12}, labOf(\psi) = labOf(e_1)$, $\Psi_{12} \downarrow_\kappa = \Psi_{12}$

(A4) $\forall ExtCoreR \in ExtCoreRs_1, EHs = getActiveHandlers(ExtCore, e_1)$,
$\forall eh \in EHs, isBlocking(eh) = \mathsf{true} \wedge \mathsf{labOf}(eh)^- = \mathsf{labOf}(e_1)$, $e_1 \notin Qof(eh)$

(A5) $(\psi'_1, \mathcal{E}''_1) = \mathsf{nextState}(\psi_1, \mathsf{noChange})$, $\Psi'_1 = \Psi_{11} :: \psi'_1$

(A6) $\Sigma'_1 = \Sigma_1[\Psi_1 \Leftarrow \Psi'_1]$

(A7) $\mathcal{E}'_1 = \mathcal{E}_1 :: \mathcal{E}''_1$

(A8) $\Sigma_2 = (\Psi_2, Tabs_2, ExtCoreRs_2, \cdots)$

sub-subcase i. $labOf(e_1) \sqsubseteq \kappa$

By Lemma 1

(1) $labOf(\psi_1) = labOf(e_1) \sqsubseteq \kappa$, $\psi_1 \downarrow_\kappa = \psi_1$

By $\Psi_1 \downarrow_\kappa \leq_c \Psi_2 \downarrow_\kappa$, which forces the projection of the two states to have the same states with regard to e_1

(2) $\exists \psi_2 \in \Psi_2 \downarrow_\kappa$, s.t. $\psi_1 \downarrow_\kappa \leq \psi_2 \downarrow_\kappa$, that is $\psi_1 \leq \psi_2$, $\psi_2 = \psi_1$.

(3) $\psi_2 \sim_b e_1$,

(4) $\Psi_2 = \Psi_{21} :: \psi_2 :: \Psi_{12}$
$\nexists \psi \in \Psi_{21}$, s.t. $\psi = \mathsf{ProcBlkEvtState}(\_, \_, e_2)$ or $\mathsf{DoneBlkEvtState}(\_, \_, e_2, \_)$

By Lemma 9

(5) $\Psi_{11} \downarrow_\kappa \leq_c \Psi_{21} \downarrow_\kappa$

By $ExtCoreRs_1 \downarrow_\kappa \leq_c ExtCoreRs_2 \downarrow_\kappa$,

(6) $ExtCoreRs_1 \downarrow_\kappa \leq ExtCoreRs_2 \downarrow_\kappa$
$ExtCoreRs_1 \downarrow_\kappa \leq_{c,e_1} ExtCoreRs_2 \downarrow_\kappa$,
$ExtCoreRs_1 \downarrow_\kappa = ExtCoreRs_{b1} :: ExtCoreRs_{nb1}$,
$ExtCoreRs_2 \downarrow_\kappa = ExtCoreRs_{b2} :: ExtCoreRs_{nb2}$
$ExtCoreRs_{b1} \leq_{B,e_1} ExtCoreRs_{b2}$

every core in $ExtCoreRs_{b1}$ and $ExtCoreRs_{b2}$ contains a blocking event handler for e_1,
no blocking event handlers for e_1 in $ExtCoreRs_{nb1}$ and $ExtCoreRs_{nb2}$

(7) $\forall ExtCoreR \in ExtCoreRs_1 \downarrow_\kappa, \exists ExtCoreR' \in ExtCoreRs_2 \downarrow_\kappa$, s.t. $ExtCoreR \downarrow_\kappa \leq ExtCoreR' \downarrow_\kappa$

By $labOf(e_1) \sqsubseteq \kappa$, and $\forall eh \in ExtCoreR, labOf(eh) = labOf(ExtCoreR)$

(8) $\forall ExtCore \in ExtCoreRs_1$, s.t. $EHs = getActiveHandlers(ExtCoreR, e_1)$,
$\exists eh \in EHs, isBlocking(eh) = \mathsf{true}$
$\wedge \mathsf{labOf}(eh)^- = \mathsf{labOf}(e_1)$
$labOf(extCoreR) \sqsubseteq \kappa$, $ExtCoreR \in ExtCoreRs_{b1}$

(9) $ExtCoreRs_{b1} \neq \cdot$

By $ExtCoreRs_{b1} \leq_{B,e_1} ExtCoreRs_{b2}$

(10) There are matching pairs of $ExtCoreR$ and $ExtCoreR'$,
s.t. $ExtCoreR = (id_{ext}, (\_, \_, EventHandlers_1 :: eh_1), \_)$
$ExtCoreR' = (id_{ext}, (\_, \_, EventHandlers_2 :: eh_2), \_)$
$eh_1 = (eventTypeOf(e_1), \_, eventQueue_1, \_, \_, \mathsf{blocking}, \_)$
$eh_2 = (eventTypeOf(e_1), \_, eventQueue_2, \_, \_, \mathsf{blocking}, \_)$
if $e_1 \notin eventQueue_1$, $e_1 \notin eventQueue_2$
$ExtCoreR \leq ExtCoreR'$

By (A4), (8), (9), (10), $e_1 = e_2$

(11) $\forall ExtCore' \in ExtCoreRs_2, EHs = getActiveHandlers(ExtCore', e_2)$,
$\forall eh \in EHs, isBlocking(eh) = \mathsf{true} \wedge labOf(eh)^- = labOf(e_2), e_2 \notin Qof(eh)$

By (4) (A2) and (11), we can apply BLOCKEDFINISHED1 to Σ_2

(12) $\Sigma_2; \mathcal{E}_2 \xrightarrow{\tau} \Sigma_2'; \mathcal{E}_2 :: \mathcal{E}_2''$

(13) $(\psi_2', \mathcal{E}_2'') = \mathsf{nextState}(\psi_2, \mathsf{noChange})$, $\Psi_2' = \Psi_{21} :: \psi_2'$

(14) $\psi_2' = \psi_1'$, $\mathcal{E}_2'' = \mathcal{E}_1''$, $labOf(\psi_2') = labOf(\psi_1') = labOf(\mathcal{E}_2'') = labOf(\mathcal{E}_1'') = labOf(e_1)$
note that we write $labOf(\mathcal{E})$ to denote the uniform label of all events in $\mathcal{E}$

(15) $\Sigma_2' = \Sigma_2[\Psi_2 \Leftarrow \Psi_2']$

By (5), (13), (14), Lemma 10

(16) $\Psi_1' \downarrow_\kappa \leq_c \Psi_2' \downarrow_\kappa$

By assumption, (14), (15), (16) and STATE

(17) $\Sigma_1' \downarrow_\kappa \leq \Sigma_2' \downarrow_\kappa$

By (A7), $\mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$

(18) $\mathcal{E}_2'' = \mathcal{E}_1''$, $\mathcal{E}_2' = \mathcal{E}_2 :: e_2'$, $\mathcal{E}_1' \downarrow_\kappa \leq \mathcal{E}_2' \downarrow_\kappa$

sub-subcase ii. $labOf(e_1) \not\sqsubseteq \kappa$

By Lemma 1, 13

(1) $labOf(\psi_1) = labOf(\psi_1') = labOf(e_1)$

(2) $\forall \psi \in \Psi_{12}, labOf(\psi) = labOf(e_1)$

(3) $labOf(e_1') = labOf(e_1)$

(4) $\psi_1 \downarrow_\kappa = \cdot$, $\mathcal{E}_1'' \downarrow_\kappa = \cdot$, $\psi_1' \downarrow_\kappa = \cdot$, $\Psi_{12} \downarrow_\kappa = \cdot$

(5) $\Psi_1' \downarrow_\kappa = \Psi_1 \downarrow_\kappa$

(6) $\Sigma_1' \downarrow_\kappa = \Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

(7) $\mathcal{E}_1' \downarrow_\kappa = \mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$

case: BLOCKFINISHED2

$$\frac{\begin{array}{l} e \text{ is a blocking event} \qquad e \sim_b \psi \qquad \Sigma = (\Psi, \mathit{Tabs}, \mathit{ExtCoreRs}, \cdots) \qquad \Psi = \Psi_1 :: \psi :: \Psi_2 \\ \nexists \psi \in \Psi_1, s.t. \psi = \mathsf{ProcBlkEvtState}(\_, \_, e) \text{ or } \mathsf{DoneBlkEvtState}(\cdots, e, \_) \\ \Psi_2 \text{ contains matching pairs of } \mathsf{ProcBlkEvtState}(\_, id_i, e) \text{ and } \mathsf{DoneBlkEvtState}(\_, id_i, e, c_i) \\ \forall \mathit{ExtCoreR} \in \mathit{ExtCoreRs}, \mathit{EHs} = \mathit{getActiveHandlers}(\mathit{ExtCoreR}, e), \\ \quad \forall eh \in \mathit{EHs}, \mathit{labOf}(eh)^- = \mathit{labOf}(e) \wedge \mathit{isBlocking}(eh) = \mathsf{true}, e \notin \mathit{Qof}(eh) \\ \mathit{getLastInstalledReturn}(\Psi_2) = (\kappa, id_j, c_j) \text{ s.t. } c_j \neq 0 \\ (\psi', \mathcal{E}_1) = \mathsf{nextState}(\psi, \mathsf{Blocked}(c_j)) \qquad \Sigma' = \Sigma[\Psi \Leftarrow \Psi_1 :: \psi'] \end{array}}{\Sigma; \mathcal{E} \xrightarrow{\tau} \Sigma'; \mathcal{E} :: \mathcal{E}_1} \text{BLOCKFINISHED2}$$

By assumptions and Lemma 1

(A1) $\Sigma_1 = (\Psi_1, \mathit{Tabs}_1, \mathit{ExtCoreRs}_1, \cdots)$, $\Psi_1 = \Psi_{11} :: \psi_1 :: \Psi_{12}$

(A2) $\psi_1 \sim_b e_1$, e_1 is a blocking event, $\mathit{labOf}(\psi_1) = \mathit{labOf}(e_1)$
$\nexists \psi \in \Psi_{11}, s.t. \psi = \mathsf{ProcBlkEvtState}(\_, \_, e_1)$ or $\mathsf{DoneBlkEvtState}(\_, \_, e_1, \_)$
Ψ_{12} contains matching pairs of $\mathsf{ProcBlkEvtState}(\_, id_j, e_1)$ and $\mathsf{DoneBlkEvtState}(\_, id_j, e_1, c_j)$

(A3) $\mathit{labOf}(\psi_1) = \mathit{labOf}(e_1)$

(A4) $\forall \mathit{ExtCoreR} \in \mathit{ExtCoreRs}_1, \mathit{EHs} = \mathit{getActiveHandlers}(\mathit{ExtCoreR}, e_1)$,
$\forall eh \in \mathit{EHs}$, $\mathit{labOf}(eh)^- = \mathit{labOf}(e_1) \wedge \mathit{isBlocking}(eh) = \mathsf{true}$, $e_1 \notin \mathit{Qof}(eh)$

(A5) $\forall \psi \in \Psi_{12}, \mathit{labOf}(\psi) = \mathit{labOf}(e_1)$

(A6) $\mathit{getLastInstalledReturn}(\Psi_{12}) = (id_{j1}, c_{j1})$ s.t. $c_{j1} \neq 0$

(A7) $(\psi_1', \mathcal{E}_1'') = \mathsf{nextState}(\psi_1, \mathsf{Blocked}(c_j))$

(A8) $\Psi_1' = \Psi_{11} :: \psi_1'$

(A9) $\Sigma_1' = \Sigma_1[\Psi_1 \Leftarrow \Psi_1']$

(A10) $\mathcal{E}_1' = \mathcal{E}_1 :: \mathcal{E}_1''$

(A11) $\Sigma_2 = (\Psi_2, \mathit{Tabs}_2, \mathit{ExtCoreRs}_2, \cdots)$

sub-subcase i. $\mathit{labOf}(e_1) \sqsubseteq \kappa$

By Lemma 1

(1) $\mathit{labOf}(\psi_1) = \mathit{labOf}(e_1) \sqsubseteq \kappa$
$\forall \psi \in \Psi_{12}$, $\mathit{labOf}(\psi) = \mathit{labOf}(e_1) \sqsubseteq \kappa$
$\psi_1 \downarrow_\kappa = \psi_1$, $\Psi_{12} \downarrow_\kappa = \Psi_{12}$

By $\Psi_1 \downarrow_\kappa \leq_c \Psi_2 \downarrow_\kappa$, (1)

(2) $\exists \psi_2 \in \Psi_2 \downarrow_\kappa$, s.t. $\psi_1 \downarrow_\kappa \leq \psi_2 \downarrow_\kappa$, that is $\psi_1 \leq \psi_2$, $\psi_2 = \psi_1$.

(3) $\psi_2 \sim_b e_2$, $e_1 = e_2$, e_2 is a blocking event

(4) $\Psi_2 = \Psi_{21} :: \psi_2 :: \Psi_{12}$
$\nexists \psi \in \Psi_{21}$, s.t. $\psi = \mathsf{ProcBlkEvtState}(\_, \_, e_2)$ or $\mathsf{DoneBlkEvtState}(\_, \_, e_2, \_)$

By Lemma 9

(5) $\Psi_{11} \downarrow_\kappa \leq_c \Psi_{21} \downarrow_\kappa$

(6) $\Psi_{22} = \Psi_{12}$,
$\mathit{getLastInstalledReturn}(\Psi_{22}) = (id_{j2}, c_{j2})$ s.t. $c_{j2} \neq 0$
$c_{j2} = c_{j1}$

By $\mathit{ExtCoreRs}_1 \downarrow_\kappa \leq_c \mathit{ExtCoreRs}_2 \downarrow_\kappa$,

(7) $\mathit{ExtCoreRs}_1 \downarrow_\kappa \leq \mathit{ExtCoreRs}_2 \downarrow_\kappa$
$\mathit{ExtCoreRs}_1 \downarrow_\kappa \leq_{c,e_1} \mathit{ExtCoreRs}_2 \downarrow_\kappa$,
$\mathit{ExtCoreRs}_1 \downarrow_\kappa = \mathit{ExtCoreRs}_{b1} :: \mathit{ExtCoreRs}_{nb1}$,
$\mathit{ExtCoreRs}_2 \downarrow_\kappa = \mathit{ExtCoreRs}_{b2} :: \mathit{ExtCoreRs}_{nb2}$
$\mathit{ExtCoreRs}_{b1} \leq_{B,e_1} \mathit{ExtCoreRs}_{b2}$
every core in $\mathit{ExtCoreRs}_{b1}$ and $\mathit{ExtCoreRs}_{b2}$ contains a blocking event handler for e_1,
no blocking event handlers for e_1 in $\mathit{ExtCoreRs}_{nb1}$ and $\mathit{ExtCoreRs}_{nb2}$

(8) $\forall ExtCoreR \in ExtCoreRs_1 \downarrow_\kappa, \exists ExtCoreR' \in ExtCoreRs_2 \downarrow_\kappa$, s.t. $ExtCoreR \downarrow_\kappa \leq ExtCoreR' \downarrow_\kappa$

By $labOf(e_1) \sqsubseteq \kappa$, and $\forall eh \in ExtCoreR, labOf(eh) = labOf(ExtCoreR)$

(9) $\forall ExtCore \in ExtCoreRs_1$, s.t. $EHs = getActiveHandlers(ExtCoreR, e_1)$,
$\exists eh \in EHs, isBlocking(eh) = \mathsf{true}$
$\wedge \mathsf{labOf}(eh)^- = \mathsf{labOf}(e_1)$
$labOf(extCoreR) \sqsubseteq \kappa, ExtCoreR \in ExtCoreRs_{b1}$

(10) $ExtCoreRs_{b1} \neq \cdot$

By $ExtCoreRs_{b1} \leq_{B,e_1} ExtCoreRs_{b2}$

(11) There are matching pairs of $ExtCoreR$ and $ExtCoreR'$,
s.t. $ExtCoreR = (id_{ext}, (\_, \_, EventHandlers_1 :: eh_1), \_)$
$ExtCoreR' = (id_{ext}, (\_, \_, EventHandlers_2 :: eh_2), \_)$
$eh_1 = (eventTypeOf(e_1), \_, eventQueue_1, \_, \_, \mathsf{blocking}, \_)$
$eh_2 = (eventTypeOf(e_1), \_, eventQueue_2, \_, \_, \mathsf{blocking}, \_)$
if $e_1 \notin eventQueue_1$, $e_1 \notin eventQueue_2$
$ExtCoreR \leq ExtCoreR'$

By (A4), (9), (10), (11), $e_1 = e_2$

(12) $\forall ExtCore' \in ExtCoreRs_2, EHs = getActiveHandlers(ExtCore', e_2)$,
$\forall eh \in EHs, isBlocking(eh) = \mathsf{true} \wedge labOf(eh)^- = labOf(e_2), e_2 \notin Qof(eh)$

By (A2), (4), (12), apply BLOCKEDFINISHED2 to Σ_2

(13) $(\psi_2', \mathcal{E}_2'') = \mathsf{nextState}(\psi_2, \mathsf{Blocked}(c_{j2}))$

(14) $\psi_2' = \psi_1', \mathcal{E}_2'' = \mathcal{E}_1''$

(15) $\Psi_2' = \Psi_{21} :: \psi_2'$

(16) $\Sigma_2' = \Sigma_2[\Psi_2 \Leftarrow \Psi_2']$

(17) $\Sigma_2; \mathcal{E}_2 \xrightarrow{\tau} \Sigma_2'; \mathcal{E}_2'$

(18) $\mathcal{E}_2' = \mathcal{E}_2 :: \mathcal{E}_2''$

By (5), (14), Lemma 10

(19) $\Psi_1' \downarrow_\kappa \leq_c \Psi_2' \downarrow_\kappa$

By assumption, (A9), (16), (19) and STATE

(20) $\Sigma_1' \downarrow_\kappa \leq \Sigma_2' \downarrow_\kappa$

By $\mathcal{E}_1 \leq \mathcal{E}_2$, (A10), (18), (22)

(21) $\mathcal{E}_1' \downarrow_\kappa \leq \mathcal{E}_2' \downarrow_\kappa$

sub-subcase ii. $labOf(e_1) \not\sqsubseteq \kappa$

By Lemma 1, 13

(1) $labOf(\psi_1) = labOf(\psi_1') = labOf(e_1)$

(2) $\forall \psi \in \Psi_{12}, labOf(\psi) = labOf(e_1)$

(3) $labOf(e_1') = labOf(e_1)$

(4) $\psi_1 \downarrow_\kappa = \cdot, \mathcal{E}_1'' \downarrow_\kappa = \cdot, \psi_1' \downarrow_\kappa = \cdot, \Psi_{12} \downarrow_\kappa = \cdot$

(5) $\Psi_1' \downarrow_\kappa = \Psi_1 \downarrow_\kappa$

(6) $\Sigma_1' \downarrow_\kappa = \Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

(7) $\mathcal{E}_1' \downarrow_\kappa = \mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$

case: EVENTDEQUEUECS1

$$\dfrac{\begin{array}{l}\Sigma_{ctx}(id) = (\Psi, \mathit{Tabs}_{\mathbf{ctx}}(id), \cdots) \qquad t_{\mathbf{ctx}}(id) \in \mathit{Tabs}_{\mathbf{ctx}}(id) \\ t_{\mathbf{ctx}}(id) = (\cdots, \mathit{Docs}_{\mathbf{ctx}}(id), \cdots) \qquad \mathit{Doc}_{\mathbf{ctx}}(id) \in \mathit{Docs}_{\mathbf{ctx}}(id) \\ \mathit{Doc}_{\mathbf{ctx}}(id) = (\cdots, \mathit{DocCSs}_{\mathbf{ctx}}(id), \cdots) \qquad \mathit{DocCS}_{\mathbf{ctx}}(id) \in \mathit{DocCSs}_{\mathbf{ctx}}(id) \\ \mathit{DocCS}_{\mathbf{ctx}}(id) = (\_, id, \_, \llbracket\,\rrbracket, \mathit{EventHandlers} :: \mathit{eh}_{\mathbf{ctx}}, \ell) \\ \mathit{eh}_{\mathbf{ctx}} = (\cdot, \mathit{eventType}, x.\mathit{cmd}, e :: \mathcal{E}_1, \llbracket\rrbracket, id_e, \mathit{BlockingFlag}, \mathit{return}) \\ \mathit{labOf}(e) \not\sqsubseteq \ell^* \qquad \mathit{eh}'_{\mathbf{ctx}} = (\cdot, \mathit{eventType}, x.\mathit{cmd}, \mathcal{E}_1, \llbracket\rrbracket, id_e, \mathit{BlockingFlag}, \mathit{return}) \\ \mathit{DocCS}'_{\mathbf{ctx}}(id) = \mathit{DocCS}_{\mathbf{ctx}}(id)[\mathit{eh}_{\mathbf{ctx}} \Leftarrow \mathit{eh}'_{\mathbf{ctx}}] \\ \mathit{DocCSs}'_{\mathbf{ctx}}(id) = \mathit{DocCSs}_{\mathbf{ctx}}(id)[\mathit{DocCS}_{\mathbf{ctx}}(id) \Leftarrow \mathit{DocCS}'_{\mathbf{ctx}}(id)] \\ \mathit{Doc}'_{\mathbf{ctx}}(id) = \mathit{Doc}_{\mathbf{ctx}}(id)[\mathit{DocCSs}_{\mathbf{ctx}}(id) \Leftarrow \mathit{DocCSs}'_{\mathbf{ctx}}(id)] \\ t'_{\mathbf{ctx}}(id) = t_{\mathbf{ctx}}(id)[\mathit{Doc}_{\mathbf{ctx}}(id) \Leftarrow \mathit{Doc}'_{\mathbf{ctx}}(id)] \\ \Sigma'_{ctx}(id) = \Sigma_{ctx}(id)[t_{\mathbf{ctx}}(id) \Leftarrow t'_{\mathbf{ctx}}(id)]\end{array}}{\Sigma_{ctx}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}; \mathcal{E} \xrightarrow{\tau} \Sigma'_{ctx}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}; \mathcal{E}} \text{ EVENTDEQUEUECS1}$$

Let $\Sigma_1 = (\Psi_1, \mathit{Tabs}_1, \mathit{ExtCoreRs}_1, \mathit{progInjCSs}_1, \mathit{Exts}_1, \mathit{Cookies}_1, \mathit{MBookmarks}_1, \mathit{histories}_1, \mathit{UI}_1)$
$\Sigma_2 = (\Psi_2, \mathit{Tabs}_2, \mathit{ExtCoreRs}_2, \mathit{progInjCSs}_2, \mathit{Exts}_2, \mathit{Cookies}_2, \mathit{MBookmarks}_2, \mathit{histories}_2, \mathit{UI}_2)$

By assumptions

(A1) $\Sigma_1 = \Sigma_{ctx}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}$

(A2) $\mathit{Tabs}_1 = \mathit{Tabs}''_1 :: \mathit{Tab}_1$

(A3) $\mathit{Tab}_1 = \mathit{Tab}_{\mathbf{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} = (\mathit{TabID}_1, \mathit{Docs} :: \mathit{Doc}_{\mathbf{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}, \mathit{url}_1, \mathit{EventHandlers}_1, \ell_1)$

(A4) $\mathit{Doc}_{\mathbf{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} = (id_d, \mathit{url}, \mathit{nodes}, \mathit{DocCSs} :: \mathit{DocCS}_{\mathbf{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}, \ell_d)$

(A5) $\mathit{DocCS}_{\mathbf{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} = (id_{ext}, id, id_t, \llbracket\Gamma\rrbracket, \mathit{cmd}, \mathit{EventHandlers} :: \mathit{eh}_{\mathbf{ctx}}\llbracket\mathsf{skip}\rrbracket, \ell)$

(A6) $\mathit{eh}_{\mathbf{ctx}} = (\cdot, \mathit{eventType}, x.\mathit{cmd}, e_1 :: \mathcal{E}''_1, \llbracket\rrbracket, id_e, \mathit{BlockingFlag}, \mathit{return})$

(A7) $\mathit{eh}'_{\mathbf{ctx}} = (\cdot, \mathit{eventType}, x.\mathit{cmd}, \mathcal{E}''_1, \llbracket\rrbracket, id_e, \mathit{BlockingFlag}, \mathit{return})$

(A8) $\mathit{DocCS}'_{\mathbf{ctx}}(id) = \mathit{DocCS}_{\mathbf{ctx}}(id)[\mathit{eh}_{1\mathbf{ctx}} \Leftarrow \mathit{eh}'_{1\mathbf{ctx}}]$

(A9) $\mathit{Doc}'_{\mathbf{ctx}}(id) = \mathit{Doc}_{\mathbf{ctx}}(id)[\mathit{DocCS}_{\mathbf{ctx}}(id) \Leftarrow \mathit{DocCS}'_{\mathbf{ctx}}(id)]$

(A10) $\mathit{Tab}'_{1\mathbf{ctx}}(id) = \mathit{Tab}_{1\mathbf{ctx}}(id)[\mathit{Doc}_{\mathbf{ctx}}(id) \Leftarrow \mathit{Doc}'_{\mathbf{ctx}}(id)]$

(A11) $\mathit{Tabs}'_{1\mathbf{ctx}}(id) = \mathit{Tabs}_{1\mathbf{ctx}}(id)[\mathit{Tab}_{1\mathbf{ctx}}(id) \Leftarrow \mathit{Tab}'_{1\mathbf{ctx}}(id)]$

(A12) $\Sigma'_1 = \Sigma'_{ctx}(id) = \Sigma_{ctx}(id)[\mathit{Tabs}_{1\mathbf{ctx}}(id) \Leftarrow \mathit{Tabs}'_{1\mathbf{ctx}}(id)]$

(A13) $\mathit{labOf}(e_1) \not\sqsubseteq \ell^*$

(A14) $\mathcal{E}'_1 = \mathcal{E}_1$

sub-subcase i. $\ell^- \sqsubseteq \kappa$ and $\mathit{labOf}(e_1) \sqsubseteq \kappa$

By DS1, EH, EVENT1

(1) $\mathit{DocCS}_{\mathbf{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} \downarrow_\kappa = (id_{ext}, id, id_t, \llbracket\Gamma\rrbracket, \mathit{cmd}, \mathit{EventHandlers} \downarrow_\kappa :: \mathit{eh}_{\mathbf{ctx}}\llbracket\mathsf{skip}\rrbracket \downarrow_\kappa, \ell)$

(2) $\mathit{eh}_{\mathbf{ctx}}\llbracket\mathsf{skip}\rrbracket \downarrow_\kappa = (\cdot, \mathit{eventType}, x.\mathit{cmd}, e_1 :: \mathcal{E}''_1 \downarrow_\kappa, \llbracket\mathsf{skip}\rrbracket, id_e, \mathit{BlockingFlag}, \mathit{return})$

By definition

(3) $\mathit{DocCS}_{\mathbf{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} \downarrow_\kappa \in \mathit{Tab}_1 \downarrow_\kappa$

By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

(4) $\mathit{Tabs}_2 = \mathit{Tabs}''_2 :: \mathit{Tab}_2$ and $\mathit{Tabs}''_1 \downarrow_\kappa \leq \mathit{Tabs}''_2 \downarrow_\kappa$ and $\mathit{Tab}_1 \downarrow_\kappa \leq \mathit{Tab}_2 \downarrow_\kappa$

By definitions and (4)

(5) exists $\mathit{DocCSs}_2 = \mathit{DocCSs}'_2 :: \mathit{DocCS}_2$, $\mathit{DocCSs}_2 \in \mathit{Doc}_2$, $\mathit{Doc}_2 \in \mathit{Docs}_2$, $\mathit{Docs}_2 \in \mathit{Tab}_2$
s.t. $\mathit{DocCS}_{\mathbf{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} \downarrow_\kappa \leq \mathit{DocCS}_2 \downarrow_\kappa$
$(\mathit{DocCS}_{\mathbf{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} :: \mathit{DocCSs}) \downarrow_\kappa \leq \mathit{DocCSs}_2 \downarrow_\kappa$
$\mathit{Doc}_{\mathbf{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} \downarrow_\kappa \leq \mathit{Doc}_2 \downarrow_\kappa$
$(\mathit{Doc}_{\mathbf{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} :: \mathit{Docs}) \downarrow_\kappa \leq \mathit{Docs}_2 \downarrow_\kappa$

By (5), DOCCS

(6) $\mathit{DocCS}_2 \downarrow_\kappa = (id_{ext}, id, id_t, \llbracket\Gamma\rrbracket, \mathit{cmd}, \mathit{EventHandlers}_2 \downarrow_\kappa :: \mathit{eh}_2 \downarrow_\kappa, \ell)$
where $\mathit{EventHandlers} \downarrow_\kappa \leq \mathit{EventHandlers}_2 \downarrow_\kappa$, $\mathit{eh}_{1\mathbf{ctx}}\llbracket\mathsf{skip}\rrbracket \downarrow_\kappa \leq \mathit{eh}_2 \downarrow_\kappa$

By (6)

(7) $eh_2 = (\cdot, eventType, x.cmd, e_2 :: \mathcal{E}''_2, \llbracket \mathsf{skip} \rrbracket, id_e, BlockingFlag, return)$
where $e_1 \leq e_2$, namely $e_1 = e_2$ and $\mathcal{E}''_1 \downarrow_\kappa \leq \mathcal{E}''_2 \downarrow_\kappa$.

(8) $eh_2 = eh_{2\,ctx2} \llbracket \mathsf{skip} \rrbracket$

(9) $DocCS_2 = DocCS_{2\,ctx2} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id}$
$DocCSs_2 = DocCSs_{2\,ctx2} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id}$
$Tab_2 = Tab_{2\,ctx2} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id}$
$Tabs_2 = Tabs_{2\,ctx2} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id}$
$\Sigma_2 = \Sigma_{2\,ctx2} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id}$

Apply EVENTDEQUEUECS1 to Σ_2

(11) $eh_2{}'_{ctx2} = (\cdot, eventType, x.cmd, \mathcal{E}''_2, \llbracket \mathsf{skip} \rrbracket, id_e, BlockingFlag, return)$

(12) $eh_1{}'_{\mathit{ctx}} \llbracket \mathsf{skip} \rrbracket \downarrow_\kappa \leq eh_2{}'_{ctx2} \llbracket \mathsf{skip} \rrbracket \downarrow_\kappa$

(13) $DocCS_2{}'_{ctx2}(id) = DocCS_{2\,ctx2}(id)[eh_{2\,ctx2} \Leftarrow eh_2{}'_{ctx2}]$

(14) $Doc_2{}'_{ctx2}(id) = Doc_{2\,ctx2}(id)[DocCSs_{2\,ctx2}(id) \Leftarrow DocCSs_2{}'_{ctx2}(id)]$

(15) $Tab_2{}'_{ctx2}(id) = Tab_{2\,ctx2}(id)[Doc_{2\,ctx2}(id) \Leftarrow Doc_2{}'_{ctx2}(id)]$

(16) $Tabs_2{}'_{ctx2}(id) = Tabs_{2\,ctx2}(id)[Tab_{2\,ctx2}(id) \Leftarrow Tab_2{}'_{ctx2}(id)]$

(17) $\Sigma'_{ctx2}(id) = \Sigma_{ctx2}(id)[Tabs_{2\,ctx2}(id) \Leftarrow Tabs_2{}'_{ctx2}(id)]$

(18) $\Sigma_{ctx2} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id}; \mathcal{E}_2 \xrightarrow{\tau} \Sigma'_{ctx2} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id}; \mathcal{E}_2, \mathcal{E}'_2 = \mathcal{E}_2$

By (4), (5), (12), (13), (14), (15), (16), and Lemma 6

(19) $DocCS'_{ctx}(id) \downarrow_\kappa \leq DocCS_2{}'_{ctx2}(id) \downarrow_\kappa$

(20) $DocCSs \downarrow_\kappa :: DocCS'_{ctx}(id) \downarrow_\kappa \leq DocCSs_2{}'_{ctx2}(id) \downarrow_\kappa$

(21) $Doc'_{\mathit{ctx}}(id) \downarrow_\kappa \leq Doc_2{}'_{ctx2}(id) \downarrow_\kappa$

(22) $Tab_{1\mathit{ctx}}(id)' \downarrow_\kappa \leq Tab_2{}'_{ctx2}(id) \downarrow_\kappa$

(23) $Tabs_{1\mathit{ctx}}(id)' \downarrow_\kappa \leq Tabs_2{}'_{ctx2}(id) \downarrow_\kappa$

By assumption, $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$, $\Sigma_1 = \Sigma_{ctx} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id}$, $\Sigma_2 = \Sigma_{ctx2} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id}$, (15), (21), STATE

(24) $\Sigma'_{ctx} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id} \downarrow_\kappa \leq \Sigma'_{ctx2} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id} \downarrow_\kappa$

By assumption and (16)

(25) $\mathcal{E}'_1 \downarrow_\kappa = \mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}'_2 \downarrow_\kappa = \mathcal{E}_2 \downarrow_\kappa$

sub-subcase ii. $\ell^- \sqsubseteq \kappa$ and $labOf(e_1) \not\sqsubseteq \kappa$

By DS1, EH, EVENT2

(1) $DocCS_{\mathit{ctx}} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id} \downarrow_\kappa = (id_{ext}, id, id_t, \llbracket \Gamma \rrbracket, cmd, EventHandlers \downarrow_\kappa :: eh_{\mathit{ctx}} \llbracket \mathsf{skip} \rrbracket \downarrow_\kappa, \ell)$

(2) $eh_{\mathit{ctx}} \llbracket \mathsf{skip} \rrbracket \downarrow_\kappa = (\cdot, eventType, x.cmd, \mathcal{E}''_1 \downarrow_\kappa, \llbracket \mathsf{skip} \rrbracket, id_e, BlockingFlag, return)$

(3) $eh'_{\mathit{ctx}} \llbracket \mathsf{skip} \rrbracket \downarrow_\kappa = (\cdot, eventType, x.cmd, \mathcal{E}''_1 \downarrow_\kappa, \llbracket \mathsf{skip} \rrbracket, id_e, BlockingFlag, return)$

(4) $eh'_{\mathit{ctx}} \llbracket \mathsf{skip} \rrbracket \downarrow_\kappa = eh_{\mathit{ctx}} \llbracket \mathsf{skip} \rrbracket \downarrow_\kappa$

By (4), (A8), (A9), (A10), (A11), (A12), Lemma 6 and STATE

(5) $DocCS'_{\mathit{ctx}}(id) \downarrow_\kappa \leq DocCS_{\mathit{ctx}}(id) \downarrow_\kappa$

(6) $DocCSs'_{\mathit{ctx}}(id) \downarrow_\kappa \leq DocCSs_{\mathit{ctx}}(id) \downarrow_\kappa$

(7) $Doc'_{\mathit{ctx}}(id) \downarrow_\kappa \leq Doc_{\mathit{ctx}}(id) \downarrow_\kappa$

(8) $Tab_1{}'_{\mathit{ctx}}(id) \downarrow_\kappa \leq Tab_{1\mathit{ctx}}(id) \downarrow_\kappa$

(9) $Tabs_1{}'_{\mathit{ctx}}(id) \downarrow_\kappa \leq Tabs_{1\mathit{ctx}}(id) \downarrow_\kappa$

(10) $\Sigma'_1 \downarrow_\kappa \leq \Sigma_1 \downarrow_\kappa$

By (10), $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$,

(11) $\Sigma'_1 \downarrow_\kappa \leq \Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

By (A14), $\mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$

(12) $\mathcal{E}'_1 \downarrow_\kappa = \mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$

sub-subcase iii. $\ell^- \not\sqsubseteq \kappa$

By DS2

(1) $DocCS_{\mathit{ctx}} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id} \notin \Sigma_1 \downarrow_\kappa$

By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$, $\Sigma_1 = \Sigma_{ctx}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}$, and EVENTDEQUEUECS1 does not change $\Sigma_{ctx}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}$ except $DocCS_{\mathit{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}$, $DocCS'_{\mathit{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} \notin \Sigma'_1 \downarrow_\kappa$

(2) $\Sigma'_{ctx}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} \downarrow_\kappa = \Sigma_{ctx}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

By assumption

(3) $\mathcal{E}'_1 = \mathcal{E}_1, \mathcal{E}'_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$

case: EVENTDEQUEUECS2

$$\dfrac{\begin{array}{l}\Sigma_{ctx}(id) = (\Psi, \mathit{Tabs}_{\mathit{ctx}}(id), \cdots) \qquad t_{\mathit{ctx}}(id) \in \mathit{Tabs}_{\mathit{ctx}}(id) \\ t_{\mathit{ctx}}(id) = (\cdots, \mathit{Docs}_{\mathit{ctx}}(id), \cdots) \qquad \mathit{Doc}_{\mathit{ctx}}(id) \in \mathit{Docs}_{\mathit{ctx}}(id) \\ \mathit{Doc}_{\mathit{ctx}}(id) = (\cdots, \mathit{DocCSs}_{\mathit{ctx}}(id), \cdots) \qquad \mathit{DocCS}_{\mathit{ctx}}(id) \in \mathit{DocCSs}_{\mathit{ctx}}(id) \\ \mathit{DocCS}_{\mathit{ctx}}(id) = (\_, id, \_, \llbracket\ \rrbracket, \_, \mathit{EventHandlers} :: eh_{\mathit{ctx}}, \ell) \\ eh_{\mathit{ctx}} = (\cdot, \mathit{eventType}, x.cmd, e :: \mathcal{E}_1, \llbracket\rrbracket, id_e, \mathit{BlockingFlag}, \mathit{return}') \\ e = (id'_e, \mathit{eventType}, \mathit{return}, \cdots, \kappa_e) \qquad \kappa_e \sqsubseteq \ell^* \\ \ell' = \kappa_e \rhd_{tnt} \ell \qquad eh' = (\cdot, \mathit{eventType}, x.cmd, \mathcal{E}_1, \llbracket\rrbracket, id'_e, \mathit{BlockingFlag}, \mathit{return}) \\ \mathit{DocCS}'_{\mathit{ctx}}(id) = \mathit{DocCS}_{\mathit{ctx}}(id)[eh_{\mathit{ctx}} \Leftarrow eh'_{\mathit{ctx}}][\ell \Leftarrow \ell'] \\ \mathit{DocCSs}'_{\mathit{ctx}}(id) = \mathit{DocCSs}_{\mathit{ctx}}(id)[\mathit{DocCS}_{\mathit{ctx}}(id) \Leftarrow \mathit{DocCS}'_{\mathit{ctx}}(id)] \\ \mathit{Doc}'_{\mathit{ctx}}(id) = \mathit{Doc}_{\mathit{ctx}}(id)[\mathit{DocCS}_{\mathit{ctx}}(id) \Leftarrow \mathit{DocCS}'_{\mathit{ctx}}(id)] \\ t'_{\mathit{ctx}}(id) = t_{\mathit{ctx}}(id)[\mathit{Doc}_{\mathit{ctx}}(id) \Leftarrow \mathit{Doc}'_{\mathit{ctx}}(id)] \\ \Sigma'_{ctx}(id) = \Sigma_{ctx}(id)[t_{\mathit{ctx}}(id) \Leftarrow t'_{\mathit{ctx}}(id)]\end{array}}{\Sigma_{ctx}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}; \mathcal{E} \xrightarrow{updLab(e, \ell'^-)} \Sigma'_{ctx}\llbracket \Gamma, cmd[e/x] \rrbracket_{id}; \mathcal{E}} \text{ EVENTDEQUEUECS2}$$

Let $\Sigma_1 = (\Psi_1, \mathit{Tabs}_1, \mathit{ExtCoreRs}_1, \mathit{progInjCSs}_1, \mathit{Exts}_1, \mathit{Cookies}_1, \mathit{MBookmarks}_1, \mathit{histories}_1, \mathit{UI}_1)$
$\Sigma_2 = (\Psi_2, \mathit{Tabs}_2, \mathit{ExtCoreRs}_2, \mathit{progInjCSs}_2, \mathit{Exts}_2, \mathit{Cookies}_2, \mathit{MBookmarks}_2, \mathit{histories}_2, \mathit{UI}_2)$

By assumptions

(A1) $\Sigma_1 = \Sigma_{ctx}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}$, $\mathit{Tabs}_1 = \mathit{Tabs}''_1 :: \mathit{Tab}_1$

(A2) $\mathit{Tab}_1 = \mathit{Tab}_{\mathit{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} = (\mathit{TabID}_1, \mathit{Docs} :: \mathit{Doc}_{\mathit{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}, \mathit{url}_1, \mathit{EventHandlers}_1, \ell_1)$

(A3) $\mathit{Doc}_{\mathit{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} = (id_d, \mathit{url}, \mathit{nodes}, \mathit{DocCSs} :: \mathit{DocCS}_{\mathit{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}, \ell_d)$

(A4) $\mathit{DocCS}_{\mathit{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} = (id_{ext}, id, id_t, \llbracket \Gamma \rrbracket, cmd, \mathit{EventHandlers} :: eh_{\mathit{ctx}}\llbracket \mathsf{skip} \rrbracket, \ell)$

(A5) $e_1 = (id'_e, \mathit{eventType}, \mathit{return}', \cdots, \kappa_e)$

(A6) $eh_{\mathit{ctx}}\llbracket \mathsf{skip} \rrbracket = (\cdot, \mathit{eventType}, x.cmd, e_1 :: \mathcal{E}''_1, \llbracket \mathsf{skip} \rrbracket, id_e, \mathit{BlockingFlag}, \mathit{return})$

(A7) $\kappa_e \sqsubseteq \ell^*$, $\ell' = \kappa_e \rhd_{tnt} \ell$

(A8) $eh'_{\mathit{ctx}} = (\cdot, \mathit{eventType}, x.cmd, \mathcal{E}''_1, \llbracket cmd[e/x] \rrbracket, {id_e}', \mathit{BlockingFlag}, \mathit{return}')$

(A9) $\mathit{DocCS}'_{\mathit{ctx}}(id) = \mathit{DocCS}_{\mathit{ctx}}(id)[eh_{\mathit{ctx}} \Leftarrow eh'_{\mathit{ctx}}][\ell \Leftarrow \ell']$

(A10) $\mathit{Doc}'_{\mathit{ctx}}(id) = \mathit{Doc}_{\mathit{ctx}}(id)[\mathit{DocCS}_{\mathit{ctx}}(id) \Leftarrow \mathit{DocCS}'_{\mathit{ctx}}(id)]$

(A11) $\mathit{Tab}_1{}'_{\mathit{ctx}}(id) = \mathit{Tab}_{1\mathit{ctx}}(id)[\mathit{Doc}_{\mathit{ctx}}(id) \Leftarrow \mathit{Doc}'_{\mathit{ctx}}(id)]$

(A12) $\mathit{Tabs}_1{}'_{\mathit{ctx}}(id) = \mathit{Tabs}_{1\mathit{ctx}}(id)[\mathit{Tab}_{1\mathit{ctx}}(id) \Leftarrow \mathit{Tab}_1{}'_{\mathit{ctx}}(id)]$

(A13) $\Sigma'_1 = \Sigma'_{ctx}(id) = \Sigma_{ctx}(id)[\mathit{Tabs}_{1\mathit{ctx}}(id) \Leftarrow \mathit{Tabs}_1{}'_{\mathit{ctx}}(id)]$

(A14) $\mathcal{E}'_1 = \mathcal{E}_1$

sub-subcase i. $\ell^- \sqsubseteq \kappa$, $\kappa_e \sqsubseteq \kappa$, $updLab(e, \ell'^-) \downarrow_\kappa = updLab(e, \ell'^-)$

By DS1,

(1) $\mathit{DocCS}_{\mathit{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} \downarrow_\kappa = (id_{ext}, id, id_t, \llbracket \Gamma \rrbracket, cmd, \mathit{EventHandlers} \downarrow_\kappa :: eh_{\mathit{ctx}}\llbracket \mathsf{skip} \rrbracket \downarrow_\kappa, \ell)$

By definition

(2) $\mathit{DocCS}_{\mathit{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} \downarrow_\kappa \in \mathit{Tab}_1 \downarrow_\kappa$

By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

(3) $\mathit{Tabs}_2 = \mathit{Tabs}''_2 :: \mathit{Tab}_2$ and $\mathit{Tabs}''_1 \downarrow_\kappa \leq \mathit{Tabs}''_2 \downarrow_\kappa$ and $\mathit{Tab}_1 \downarrow_\kappa \leq \mathit{Tab}_2 \downarrow_\kappa$

By definitions and (3)

(4) exists $\mathit{DocCS}_2 \in \mathit{DocCSs}_2$, $\mathit{DocCSs}_2 \in \mathit{Doc}_2$, $\mathit{Doc}_2 \in \mathit{Docs}_2$, $\mathit{Docs}_2 \in \mathit{Tab}_2$

s.t. $DocCS_{\mathit{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} \downarrow_\kappa \leq DocCS_2 \downarrow_\kappa$
$(DocCS_{\mathit{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} :: DocCSs) \downarrow_\kappa \leq DocCSs_2 \downarrow_\kappa$
$Doc_{\mathit{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} \downarrow_\kappa \leq Doc_2 \downarrow_\kappa$
$(Doc_{\mathit{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} :: Docs) \downarrow_\kappa \leq Docs_2 \downarrow_\kappa$

By (4), DocCS

(5) $DocCS_2 = (id_{ext}, id, id_t, \llbracket\Gamma\rrbracket, cmd, EventHandlers_2 :: eh_2, \ell)$
where $EventHandlers \downarrow_\kappa \leq EventHandlers_2 \downarrow_\kappa$, $eh \downarrow_\kappa \leq eh_2 \downarrow_\kappa$

By (5)

(6) $eh_2 = (\cdot, eventType, x.cmd, e_2 :: \mathcal{E}''_2, \llbracket\mathsf{skip}\rrbracket, id_e, BlockingFlag, return)$
where $e_1 \leq e_2$, namely $e_1 = e_2$ and $\mathcal{E}''_1 \downarrow_\kappa \leq \mathcal{E}''_2 \downarrow_\kappa$.

(7) $eh_2 = eh_{2\,ctx2}\llbracket \mathsf{skip} \rrbracket$

(8) $DocCS_2 = DocCS_{2\,ctx2}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}$
$DocCSs_2 = DocCSs_{2\,ctx2}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}$
$Tab_2 = Tab_{2\,ctx2}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}$
$Tabs_2 = Tabs_{2\,ctx2}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}$
$\Sigma_2 = \Sigma_{2\,ctx2}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}$

By assumption and (6)

(9) $labOf(e_2) = labOf(e_1)$, $labOf(e_2) \sqsubseteq \kappa$ and $labOf(e_2) \sqsubseteq \ell^*$

Apply EventDequeueCS2 to Σ_2

(10) $eh_2{}'_{ctx2}\llbracket cmd[e/x]\rrbracket = (\cdot, eventType, x.cmd, \mathcal{E}''_2, \llbracket cmd[e/x]\rrbracket, id_e{}', BlockingFlag, return')$

(11) $eh'_{\mathit{ctx}}\llbracket cmd[e/x] \rrbracket \downarrow_\kappa \leq eh_2{}'_{ctx2}\llbracket cmd[e/x] \rrbracket \downarrow_\kappa$

(12) $DocCS_2{}'_{ctx2}(id) = DocCS_{2\,ctx2}(id)[eh_{2\,ctx2} \Leftarrow eh_2{}'_{ctx2}][\ell \Leftarrow \ell']$
$\ell'^- \sqsubseteq \kappa$

(13) $Doc_2{}'_{ctx2}(id) = Doc_{2\,ctx2}(id)[DocCSs_{2\,ctx2}(id) \Leftarrow DocCSs_2{}'_{ctx2}(id)]$

(14) $Tab_2{}'_{ctx2}(id) = Tab_{2\,ctx2}(id)[Doc_{2\,ctx2}(id) \Leftarrow Doc_2{}'_{ctx2}(id)]$

(15) $Tabs_2{}'_{ctx2}(id) = Tabs_{2\,ctx2}(id)[Tab_{2\,ctx2}(id) \Leftarrow Tab_2{}'_{ctx2}(id)]$

(16) $\Sigma'_{ctx2}(id) = \Sigma_{ctx2}(id)[Tabs_{2\,ctx2}(id) \Leftarrow Tabs_2{}'_{ctx2}(id)]$

(17) $\Sigma_{ctx2}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}; \mathcal{E}_2 \overset{updLab(e_2,\ell'^-)}{\longrightarrow} \Sigma'_{ctx2}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}; \mathcal{E}_2, \mathcal{E}'_2 = \mathcal{E}_2$

By (4), (11), (12), (13), (14), (15) and Lemma 6

(18) $DocCS'_{ctx}(id) \downarrow_\kappa \leq DocCS_2{}'_{ctx2}(id) \downarrow_\kappa$

(19) $DocCSs \downarrow_\kappa :: DocCS'_{ctx}(id) \downarrow_\kappa \leq DocCSs_2{}'_{ctx2}(id) \downarrow_\kappa$

(20) $Doc'_{\mathit{ctx}}(id) \downarrow_\kappa \leq Doc_2{}'_{ctx2}(id) \downarrow_\kappa$

(21) $Tab_{1\mathit{ctx}}(id)' \downarrow_\kappa \leq Tab_2{}'_{ctx2}(id) \downarrow_\kappa$

(22) $Tabs_{1\mathit{ctx}}(id)' \downarrow_\kappa \leq Tabs_2{}'_{ctx2}(id) \downarrow_\kappa$

By assumption, $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$, $\Sigma_1 = \Sigma_{ctx}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}$, $\Sigma_2 = \Sigma_{ctx2}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}$, (15), (21), STATE

(23) $\Sigma'_{ctx}\llbracket \Gamma, cmd[e/x] \rrbracket_{id} \downarrow_\kappa \leq \Sigma'_{ctx2}\llbracket \Gamma, cmd[e/x] \rrbracket_{id} \downarrow_\kappa$

By assumption and (17)

(24) $\mathcal{E}'_1 \downarrow_\kappa = \mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}'_2 \downarrow_\kappa = \mathcal{E}_2 \downarrow_\kappa$

sub-subcase ii. $\ell^- \sqsubseteq \kappa$, $\kappa_e \not\sqsubseteq \kappa$, $updLab(e, \ell'^-) \downarrow_\kappa = \cdot$

By DS1

(1) $DocCS_{\mathit{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} \downarrow_\kappa = (id_{ext}, id, id_t, \llbracket\Gamma\rrbracket, cmd, EventHandlers \downarrow_\kappa :: eh_{\mathit{ctx}}\llbracket\mathsf{skip}\rrbracket \downarrow_\kappa, \ell)$

By (A7)

(2) $\ell'^- \not\sqsubseteq \kappa$, $updLab(e, \ell'^-) \downarrow_\kappa = \cdot$

By DS2

(3) $DocCS_{\mathit{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} \downarrow_\kappa = \cdot$

By (A10), (A11), (A12), (A13) Lemma 6 and STATE

(4) $(DocCSs :: DocCSs'_{\mathit{ctx}}(id)) \downarrow_\kappa \leq (DocCSs :: DocCSs_{\mathit{ctx}}(id)) \downarrow_\kappa$

(5) $Doc'_{\mathit{ctx}}(id) \downarrow_\kappa \leq Doc_{\mathit{ctx}}(id) \downarrow_\kappa$

(6) $\quad \mathit{Tab_1}'_{\mathit{ctx}}(id) \downarrow_\kappa \leq \mathit{Tab_1}_{\mathit{ctx}}(id) \downarrow_\kappa$

(7) $\quad \mathit{Tabs_1}'_{\mathit{ctx}}(id) \downarrow_\kappa \leq \mathit{Tabs_1}_{\mathit{ctx}}(id) \downarrow_\kappa$

(9) $\quad \Sigma_1' \downarrow_\kappa \leq \Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

By (A14)

(10) $\mathcal{E}_1' \downarrow_\kappa = \mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$

sub-subcase iii. $\ell^- \not\sqsubseteq \kappa$, $\kappa_e \not\sqsubseteq \kappa$, $\mathit{updLab}(e, \ell'^-) \downarrow_\kappa = \cdot$

By DS2

(1) $\quad \mathit{DocCS}_{\mathit{ctx}} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id} \notin \Sigma_1 \downarrow_\kappa$

By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$, $\Sigma_1 = \Sigma_{ctx} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id}$, and applying EVENTDEQUEUECS2 only changes DocCS, $\mathit{DocCS} \downarrow_\kappa = \cdot$

(2) $\quad \Sigma'_{ctx} \llbracket \Gamma, \mathit{cmd}[e/x] \rrbracket_{id} \downarrow_\kappa = \Sigma_{ctx} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id} \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

By assumption

(3) $\quad \mathcal{E}_1' = \mathcal{E}_1, \mathcal{E}_1' \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$

case: The proofs for EVENTDEQUEUECORE1 are similar to the proofs for EVENTDEQUEUECS1.

case: The proofs for EVENTDEQUEUECORE2, are similar to the proofs for EVENTDEQUEUECS2.

$$\frac{\begin{array}{l} \Sigma_{ctx}(id) = (\Psi, \mathit{Tabs}_{\mathit{ctx}}(id), \cdots) \qquad t_{\mathit{ctx}}(id) \in \mathit{Tabs}_{\mathit{ctx}}(id) \\ t_{\mathit{ctx}}(id) = (id_t, \mathit{Docs}_{\mathit{env}}(id), \mathit{url}, \mathit{EventHandlers} :: \mathit{eh}_{\mathit{ctx}}(id), \_) \\ \mathit{eh}_{\mathit{ctx}}(id) = (id, \mathit{eventType}, x.\mathit{cmd}, e :: \mathcal{E}_1, \llbracket\rrbracket, id_e, \mathit{BlockingFlag}, \mathit{return}) \\ \mathit{Doc}_{\mathit{env}}(id) \in \mathit{Docs}_{\mathit{env}}(id) \qquad \mathit{Doc}_{\mathit{env}}(id) = (\cdots, \mathit{nodes}_{\mathit{env}}(id), \cdots) \\ \mathit{node}_{\mathit{env}}(id) \in \mathit{nodes}_{\mathit{env}}(id) \qquad \mathit{node}_{\mathit{env}}(id) = (id, \_, \_, \_, \llbracket\rrbracket, \ell) \qquad \mathit{labOf}(e) \not\sqsubseteq \ell^* \\ t'_{\mathit{ctx}}(id) = t_{\mathit{ctx}}(id)[\mathit{eh}_{\mathit{ctx}}(id) \Leftarrow \mathit{eh}'_{\mathit{ctx}}(id)] \qquad \Sigma'_{ctx}(id) = \Sigma_{ctx}(id)[t_{\mathit{ctx}}(id) \Leftarrow t'_{\mathit{ctx}}(id)] \end{array}}{\Sigma_{ctx} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id}; \mathcal{E} \xrightarrow{\tau} \Sigma'_{ctx} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id}; \mathcal{E}} \text{ EVENTDEQUEUEPAGE1}$$

case: EVENTDEQUEUEPAGE1 satisfies Lemma 16.

Let $\Sigma_1 = (\Psi_1, \mathit{Tabs}_1, \mathit{ExtCoreRs}_1, \mathit{progInjCSs}_1, \mathit{Exts}_1, \mathit{Cookies}_1, \mathit{MBookmarks}_1, \mathit{histories}_1, \mathit{UI}_1)$

$\Sigma_2 = (\Psi_2, \mathit{Tabs}_2, \mathit{ExtCoreRs}_2, \mathit{progInjCSs}_2, \mathit{Exts}_2, \mathit{Cookies}_2, \mathit{MBookmarks}_2, \mathit{histories}_2, \mathit{UI}_2)$

By assumptions

(A1) $\quad \Sigma_1 = \Sigma_{ctx} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id}$, $\mathit{Tabs}_1 = \mathit{Tabs}_1'' :: \mathit{Tab}_1$

(A2) $\quad \mathit{Tab}_1 = \mathit{Tab}_{\mathit{ctx}} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id}$
$\quad\quad = (id_t, \mathit{Docs}_1'' :: \mathit{Doc}_{\mathit{env}}(id), \mathit{url}_1, \mathit{EventHandlers}_{\mathit{ctx}}(id), \ell_1)$

(A3) $\quad \mathit{EventHandler}_{\mathit{ctx}}(id) \in \mathit{EventHandlers}_{\mathit{ctx}}(id)$
$\quad \mathit{EventHandler}_{\mathit{ctx}}(id) = (id, \mathit{eventType}, x.\mathit{cmd}, e_1 :: \mathcal{E}_1'', \llbracket\rrbracket, id_e, \mathit{BlockingFlag}, \mathit{return})$

(A4) $\quad \mathit{Doc}_{\mathit{env}}(id) = (\cdots, \mathit{nodes}_{\mathit{env}}(id), \cdots)$

(A5) $\quad \mathit{node}_{\mathit{env}}(id) \in \mathit{nodes}_{\mathit{env}}(id)$

(A6) $\quad \mathit{node}_{\mathit{env}}(id) = (id, \_, \_, \_, \llbracket \Gamma \rrbracket, \ell)$

(A7) $\quad \mathit{labOf}(e_1) \not\sqsubseteq \ell^*$

(A8) $\quad \mathit{eh}'_{\mathit{ctx}}(id) = \mathit{eh}_{\mathit{ctx}}(id)[e_1 :: \mathcal{E}_1'' \Leftarrow \mathcal{E}_1'']$

(A9) $\quad \mathit{Tab}'_{\mathit{ctx}}(id) = (id_t, \mathit{Docs}(id)_{\mathit{env}}, \mathit{url}, \mathit{EventHandlers}_{\mathit{ctx}}(id), \mathcal{E}_1, \ell_1)$

(A10) $\Sigma_1' = \Sigma'_{ctx}(id) = \Sigma_{ctx}(id)[\mathit{Tab}_{\mathit{ctx}}(id) \Leftarrow \mathit{Tab}'_{\mathit{ctx}}(id)]$

(A11) $\mathcal{E}_1' = \mathcal{E}_1$

By (A8), (A9), (A10)

(1) $\quad$ The only difference between Σ_1' and Σ_1 is Σ_1 contains e_1, while Σ_1' does not.

By TAB1, TAB2, EVENT1, EVENT2

(2) $\quad \forall e_1, \Sigma_1' \downarrow_\kappa \leq \Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

By (A11)

(3) $\mathcal{E}_1' = \mathcal{E}_1 \leq \mathcal{E}_2$

case: EVENTDEQUEUEPAGE2

Let $\Sigma_1 = (\Psi_1, \mathit{Tabs}_1, \mathit{ExtCoreRs}_1, \mathit{progInjCSs}_1, \mathit{Exts}_1, \mathit{Cookies}_1, \mathit{MBookmarks}_1, \mathit{histories}_1, \mathit{UI}_1)$
$\Sigma_2 = (\Psi_2, \mathit{Tabs}_2, \mathit{ExtCoreRs}_2, \mathit{progInjCSs}_2, \mathit{Exts}_2, \mathit{Cookies}_2, \mathit{MBookmarks}_2, \mathit{histories}_2, \mathit{UI}_2)$

By assumptions

(A1) $\Sigma_1 = \Sigma_{ctx}[\![\Gamma, \mathsf{skip}]\!]_{id}$, $\mathit{Tabs}_1 = \mathit{Tabs}_1'' :: \mathit{Tab}_1$

(A2) $\mathit{Tab}_1 = \mathit{Tab}_{\mathit{ctx}}[\![\Gamma, \mathsf{skip}]\!]_{id}$
$= (id_t, \mathit{Docs}_{\mathit{env}}(id), url_1, \mathit{EventHandlers}_{\mathit{ctx}}(id), \ell_1)$

(A3) $\mathit{EventHandler}_{\mathit{ctx}}(id) \in \mathit{EventHandlers}_{\mathit{ctx}}(id)$

(A4) $\mathit{EventHandler}_{\mathit{ctx}}(id) = (id, \mathit{eventType}, x.\mathit{cmd}, e_1 :: \mathcal{E}_1, [\![\mathsf{skip}]\!], id_e, \mathit{BlockingFlag}, \mathit{return})$

(A5) $\mathit{Doc}_{\mathit{env}}(id) \in \mathit{Docs}_{\mathit{env}}(id)$, $\mathit{Doc}_{\mathit{env}}(id) = (\cdots, \mathit{nodes}_{\mathit{env}}(id), \cdots)$

(A6) $\mathit{node}_{\mathit{env}}(id) \in \mathit{nodes}_{\mathit{env}}(id)$

(A7) $\mathit{node}_{\mathit{env}}(id) = (id, \_, \mathit{nodes}_1, \_, [\![\Gamma]\!], \ell)$

(A8) $e_1 = (id_e', \mathit{eventType}, \mathit{return}', \cdots, \kappa_e)$

(A9) $\kappa_e \sqsubseteq \ell^*$

(A10) $\ell' = \kappa_e \rhd_{\mathit{tnt}} \ell$

(A11) $\mathit{EventHandler}'_{1\mathit{ctx}}(id) = (id, \mathit{eventType}, x.\mathit{cmd}, \mathcal{E}_1, [\![\mathit{cmd}[e_1/x]]\!], {id_e}', \mathit{BlockingFlag}, \mathit{return}')$

(A12) $\mathit{node}'_{\mathit{env}}(id) = \mathit{node}_{\mathit{env}}(id)[\ell \Leftarrow \ell']$, $\mathit{nodes}'_{\mathit{env}}(id) = \mathit{nodes}_{\mathit{env}}(id)[\mathit{node}_{\mathit{env}}(id) \Leftarrow \mathit{node}'_{\mathit{env}}(id)]$

(A13) $\mathit{Docs}'_{\mathit{env}}(id) = \mathit{Docs}_{\mathit{env}}(id)[\mathit{node}_{\mathit{env}}(id) \Leftarrow \mathit{node}'_{\mathit{env}}(id)]$

(A14) $\mathit{Tab}'_{\mathit{ctx}}(id) = \mathit{Tab}_{\mathit{ctx}}(id)[\mathit{Docs}(id)_{\mathit{env}} \Leftarrow \mathit{Docs}'(id)_{\mathit{env}}]$

(A15) $\Sigma_1' = \Sigma'_{ctx}(id) = \Sigma_{ctx}(id)[\mathit{Tab}_{\mathit{ctx}}(id) \Leftarrow \mathit{Tab}'_{\mathit{ctx}}(id)]$

(A16) $\mathcal{E}_1' = \mathcal{E}_1$

sub-subcase i. $\ell^- \sqsubseteq \kappa$, $\kappa_e \sqsubseteq \kappa$, $\mathit{updLab}(e_1, \ell'^-) \downarrow_\kappa = \mathit{updLab}(e_1, \ell'^-)$

By TAB1, TAB2, EVENT1, (A8)

(1) $\mathit{Tab}_{1\mathit{ctx}}[\![\Gamma, \mathsf{skip}]\!]_{id} \downarrow_\kappa = (id_t, \mathit{Docs}_{\mathit{env}}(id) \downarrow_\kappa, \_,$
$\mathsf{prune}(\mathit{EventHandlers}_{\mathit{ctx}}(id), \mathit{Docs}_{\mathit{env}}(id) \downarrow_\kappa) \downarrow_\kappa, \_)$

(2) $\mathit{Tab}_{1\mathit{ctx}}[\![\Gamma, \mathsf{skip}]\!]_{id} \downarrow_\kappa \in \Sigma_1 \downarrow_\kappa$

By (A7), NODE1, EVENT1

(3) $\mathit{node}'_{\mathit{env}}(id) \downarrow_\kappa \in \Sigma_1 \downarrow_\kappa$

(4) $id \in \mathit{Doc}_{\mathit{env}}(id) \downarrow_\kappa$

(5) $\mathit{EventHandler}_{\mathit{ctx}}(id) \downarrow_\kappa = (id, \mathit{eventType}, x.\mathit{cmd}, e_1 :: \mathcal{E}_1 \downarrow_\kappa, [\![\mathsf{skip}]\!], id_e, \mathit{BlockingFlag}, \mathit{return})$

(6) $\mathit{EventHandler}_{\mathit{ctx}}(id) \downarrow_\kappa \in \mathit{Tab}_{\mathit{ctx}}[\![\Gamma, \mathsf{skip}]\!]_{id} \downarrow_\kappa$

By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

(7) $\exists \mathit{Tab}_{2\mathit{ctx}}[\![\Gamma, \mathsf{skip}]\!]_{id} \in \Sigma_2$, s.t. $\mathit{Tab}_{\mathit{ctx}}[\![\Gamma, \mathsf{skip}]\!]_{id} \downarrow_\kappa \leq \mathit{Tab}_{2\mathit{ctx}}[\![\Gamma, \mathsf{skip}]\!]_{id} \downarrow_\kappa$

(8) $\mathit{Tab}_{2\mathit{ctx}}[\![\Gamma, \mathsf{skip}]\!]_{id} = (id_t, \mathit{Docs}_{2\mathit{env}}(id), \_, \mathit{EventHandlers}_s'' :: \mathit{EventHandler}_{2\mathit{ctx}}(id), \_)$,
where $\mathit{Docs}_1'' \downarrow_\kappa \leq \mathit{Docs}_2'' \downarrow_\kappa$, $\mathit{Doc}_{\mathit{env}}(id) \downarrow_\kappa \leq \mathit{Doc}_{2\mathit{env}2}(id) \downarrow_\kappa$, $\mathit{Doc}_{2\mathit{env}}(id) \in \mathit{Docs}_{2\mathit{env}}(id)$
$\exists \mathit{node}_2 \in \mathit{Doc}_{2\mathit{env}2}(id)$, s.t., $\mathit{node}'_{\mathit{env}}(id) \downarrow_\kappa \leq \mathit{node}_2 \downarrow_\kappa$,
$\mathit{node}_2 = \mathit{node}_{2\mathit{env}}'(id) = (id, \_, \mathit{nodes}_2, \_, [\![\Gamma]\!], \ell)$
$id \in \mathit{Doc}_{2\mathit{env}}(id) \downarrow_\kappa$
$\mathit{EventHandler}_{2\mathit{ctx}}(id) \downarrow_\kappa \in \mathit{Tab}_{2\mathit{ctx}}[\![\Gamma, \mathsf{skip}]\!]_{id} \downarrow_\kappa$
$\mathit{EventHandler}_{\mathit{ctx}}(id) \downarrow_\kappa \leq \mathit{EventHandler}_{2\mathit{ctx}}(id) \downarrow_\kappa$
$\mathit{EventHandler}_{2\mathit{ctx}}(id) \downarrow_\kappa = (id, \mathit{eventType}, x.\mathit{cmd}, e_2 :: \mathcal{E}_2, [\![\mathsf{skip}]\!], id_e, \mathit{BlockingFlag}, \mathit{return})$
$\mathcal{E}_1'' \downarrow_\kappa \leq \mathcal{E}_2'' \downarrow_\kappa$, $e_1 \downarrow_\kappa \leq e_2 \downarrow_\kappa$, that is $e_1 = e_2$

By $e_1 = e_2$, (8), apply EVENTDEQUEUEPAGE2 to Σ_2, $\Sigma_2 = \Sigma_{2\,ctx2}[\![\Gamma, \mathsf{skip}]\!]_{id}$

(9) $\mathit{EventHandler}'_{2\,ctx2}(id) = (id, \mathit{eventType}, x.\mathit{cmd}, \mathcal{E}_2, [\![\mathit{cmd}[e_2/x]]\!], {id_e}', \mathit{BlockingFlag}, \mathit{return}')$

(10) $\mathit{node}_2{}'_{\mathit{env}}(id) = \mathit{node}_{2\mathit{env}}(id)[\ell \Leftarrow \ell']$

(11) $Docs_2{}'_{env}(id) = Docs_{2env}(id)[node_{2env}(id) \Leftarrow node_2{}'_{env}(id)]$
(12) $Tab_2{}'_{ctx}(id) = Tab_{2ctx}(id)[Docs_2(id)_{env} \Leftarrow Docs_2{}'(id)_{env}]$
(13) $\Sigma'_1 = \Sigma'_{ctx}(id) = \Sigma_{ctx}(id)[Tab_{2ctx}(id) \Leftarrow Tab_2{}'_{ctx}(id)]$
(14) $\mathcal{E}'_2 = \mathcal{E}_2$
By (12), (A11), (A12), (A13), (A14), (A15), (9), (10), (11), (12), (13), Lemma 6, STATE
(17) $node'_{env}(id) \downarrow_\kappa \leq_n node'_{2env2}(id) \downarrow_\kappa$
(18) $Docs'_{env}(id) \downarrow_\kappa \leq Docs_2{}'_{env2}(id) \downarrow_\kappa$
(19) $Tab'_{ctx}(id) \downarrow_\kappa \leq Tab_2{}'_{ctx2}(id) \downarrow_\kappa$
(20) $\Sigma'_1 \downarrow_\kappa \leq \Sigma'_2 \downarrow_\kappa$
By (A16), (14), $\mathcal{E}_1 \leq \mathcal{E}_2$
(21) $\mathcal{E}'_1 \leq \mathcal{E}'_2$

sub-subcase ii. $\ell^- \sqsubseteq \kappa$, $labOf(e_1) \not\sqsubseteq \kappa$, $updLab(e_1, \ell'^-) \downarrow_\kappa = \cdot$
By (A10), $labOf(e_1) \not\sqsubseteq \kappa$
(1) $\ell'^- \not\sqsubseteq \kappa$
By (1), (A11-14), NODE, NODE1, NODE2, TAB1, TAB2, Lemma 6
(2) $node_{env}{}' \downarrow_\kappa = nodes_1 \downarrow_\kappa$
(3) $nodes_{env}(id) \downarrow_\kappa \leq_{ns} nodes'_{env}(id) \downarrow_\kappa$
(4) $Doc_{env}(id) \downarrow_\kappa \leq Doc'_{env}(id) \downarrow_\kappa$
(5) $Docs_{env}(id) \downarrow_\kappa \leq Docs'_{env}(id) \downarrow_\kappa$
(6) $\nexists id \in Docs_1{}'_{env}(id) \downarrow_\kappa$
(7) $EventHandler'_{ctx}(id) \downarrow_\kappa \notin Tab'_{ctx}(id) \downarrow_\kappa$
(8) $\mathsf{prune}(EventHandlers'_{ctx}(id), Docs'_{env}(id) \downarrow_\kappa) \downarrow_\kappa \leq \mathsf{prune}(EventHandlers_{ctx}(id), Docs_{env}(id) \downarrow_\kappa) \downarrow_\kappa$
(9) $Tab_{ctx}(id) \downarrow_\kappa \leq Tab'_{ctx}(id) \downarrow_\kappa$
By (A15), STATE
(10) $\Sigma'_1 \downarrow_\kappa \leq \Sigma_1 \downarrow_\kappa$
By (A16)
(11) $\mathcal{E}'_1 \downarrow_\kappa \leq \mathcal{E}_1 \downarrow_\kappa$

sub-subcase iii. $\ell^- \not\sqsubseteq \kappa$, $updLab(e_1, \ell'^-) \downarrow_\kappa = \cdot$
By (A5), (A6), (A7), TAB1, TAB2 DOC1, DOC2, NODE2
(1) $\nexists node = (id, \cdots) \in Doc_{env}(id) \downarrow_\kappa$
(2) $id \notin Docs_{env}(id) \downarrow_\kappa$
(3) $\mathsf{prune}(EventHandler_{ctx}(id), Docs_{env}(id) \downarrow_\kappa) \downarrow_\kappa \notin Tab_{ctx}(id) \downarrow_\kappa$
By $\ell^- \not\sqsubseteq \kappa$, (A12), TAB1, TAB2 DOC1, DOC2, NODE2
(4) $\ell'^- \not\sqsubseteq \kappa$
(5) $\nexists node = (id, \cdots) \in Doc'_{env}(id) \downarrow_\kappa$
(6) $id \notin Docs'_{env}(id) \downarrow_\kappa$
(7) $\mathsf{prune}(EventHandler_{ctx}(id), Docs'_{env}(id) \downarrow_\kappa) \downarrow_\kappa \notin Tab'_{ctx}(id) \downarrow_\kappa$
By (A11-15), (1), (3), (5), (7)
and EVENTDEQUEUEPAGE2 dose not change other components except $node_{env}(id)$ and $EventHandler_{ctx}(id)$
(8) $\Sigma'_1 \downarrow_\kappa \leq \Sigma_1 \downarrow_\kappa$
By (A15)
(9) $\mathcal{E}'_1 \downarrow_\kappa \leq \mathcal{E}_1 \downarrow_\kappa$

case: EVENTDEQUEUECOREBLOCKING1

$$\dfrac{\begin{array}{l}\Sigma_{ctx}(id) = (\Psi, Tabs, ExtCoreRs_{\mathsf{ctx}}(id), \cdots) \\ ExtCoreR_{\mathsf{ctx}}(id) \in ExtCoreRs_{\mathsf{ctx}}(id) \\ ExtCoreR_{\mathsf{ctx}}(id) = (id, (\llbracket\rrbracket, \_, EventHandlers :: eh_{\mathsf{ctx}}), \ell) \\ eh_{\mathsf{ctx}} = (\cdot, eventType, x.cmd, e :: \mathcal{E}_1, \llbracket\rrbracket, id_e, \mathsf{blocking}, return) \\ labOf(e) \neq \ell^- \qquad eh'_{\mathsf{ctx}} = (\cdot, eventType, x.cmd, \mathcal{E}_1, \llbracket\rrbracket, id_e, \mathsf{blocking}, return) \\ ExtCoreR_{\mathsf{ctx}}{}'(id) = ExtCoreR_{\mathsf{ctx}}(id)[eh_{\mathsf{ctx}} \Leftarrow eh'_{\mathsf{ctx}}] \\ ExtCoreRs_{\mathsf{ctx}}{}'(id) = ExtCoreRs_{\mathsf{ctx}}(id)[ExtCoreRs_{\mathsf{ctx}}(id) \Leftarrow ExtCoreRs_{\mathsf{ctx}}{}'(id)] \\ \Sigma'_{ctx}(id) = \Sigma_{ctx}(id)[ExtCoreRs_{\mathsf{ctx}}(id) \Leftarrow ExtCoreRs'_{\mathsf{ctx}}(id)]\end{array}}{\Sigma_{ctx}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}; \mathcal{E} \xrightarrow{\tau} \Sigma'_{ctx}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}; \mathcal{E}} \text{ EVENTDEQUEUECOREBLOCKING1}$$

Let $\Sigma_1 = (\Psi_1, Tabs_1, ExtCoreRs_1, progInjCSs_1, Exts_1, Cookies_1, MBookmarks_1, histories_1, UI_1)$
$\Sigma_2 = (\Psi_2, Tabs_2, ExtCoreRs_2, progInjCSs_2, Exts_2, Cookies_2, MBookmarks_2, histories_2, UI_2)$

By assumptions

(A1) $\Sigma_1 = \Sigma_{ctx}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}$, $ExtCoreRs_1 = ExtCoreRs'' :: ExtCoreR$

(A2) $ExtCoreR = ExtCoreR_{1\mathsf{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} = (id, (\llbracket\Gamma\rrbracket, cmd, EHs :: eh_{ctx}\llbracket\mathsf{skip}\rrbracket), \ell)$

(A3) $eh_{ctx}\llbracket\mathsf{skip}\rrbracket = (\cdot, eventType, x.cmd, \mathcal{E} :: e_1, \llbracket\mathsf{skip}\rrbracket, id_e, \mathsf{blocking}, return)$

(A4) $labOf(e_1) \neq \ell^-$

(A5) $eh'_{\mathsf{ctx}}\llbracket\mathsf{skip}\rrbracket = (\cdot, eventType, x.cmd, \mathcal{E}, \llbracket\mathsf{skip}\rrbracket, id_e, \mathsf{blocking}, return)$

(A6) $ExtCoreR_{\mathsf{ctx}}{}'(id) = ExtCoreR_{\mathsf{ctx}}(id)[eh_{\mathsf{ctx}} \Leftarrow eh'_{\mathsf{ctx}}]$

(A7) $ExtCoreRs_{\mathsf{ctx}}{}'(id) = ExtCoreRs_{\mathsf{ctx}}(id)[ExtCoreRs_{\mathsf{ctx}}(id) \Leftarrow ExtCoreRs_{\mathsf{ctx}}{}'(id)]$

(A8) $\Sigma'_{ctx}(id) = \Sigma_{ctx}(id)[ExtCoreRs_{\mathsf{ctx}}(id) \Leftarrow ExtCoreRs'_{\mathsf{ctx}}(id)]$

(A9) $\Sigma'_1 = \Sigma'_{ctx}(id)$

(A10) $\Sigma_{ctx}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}; \mathcal{E}_1 \xrightarrow{\tau} \Sigma'_{ctx}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}; \mathcal{E}'_1$, $\mathcal{E}_1 = \mathcal{E}'_1$

sub-subcase i. $\ell^- \sqsubseteq \kappa$, $e_1 \sqsubseteq \kappa$

By EXTCORE1, EH

(1) $ExtCoreR_{\mathsf{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} \downarrow_\kappa = (id, (\llbracket\Gamma\rrbracket, cmd, EHs \downarrow_\kappa :: eh_{\mathsf{ctx}}\llbracket\mathsf{skip}\rrbracket \downarrow_\kappa), \ell)$

(2) $eh_{ctx}\llbracket\mathsf{skip}\rrbracket \downarrow_\kappa = (\cdot, eventType, x.cmd, \mathcal{E} \downarrow_\kappa :: e_1, \llbracket\mathsf{skip}\rrbracket, id_e, \mathsf{blocking}, return)$

(3) $ExtCoreR_{\mathsf{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} \downarrow_\kappa \in \Sigma_1 \downarrow_\kappa$

By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

(4) $ExtCoreRs_1 \downarrow_\kappa \leq_c ExtCoreRs_2 \downarrow_\kappa$

(5) $\Psi_1 \leq_c \Psi_2$

By $ExtCoreRs_1 \downarrow_\kappa \leq_c ExtCoreRs_2 \downarrow_\kappa$

(6) $ExtCoreRs_1 \downarrow_\kappa \leq ExtCoreRs_2 \downarrow_\kappa$

(7) $\exists ExtCoreR_2$, s.t., $ExtCoreRs_2 = ExtCoreRs''_2 :: ExtCoreR_2$
$ExtCore_2 = ExtCoreR_{\mathsf{ctx}}{}'\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}$
$ExtCoreR_{\mathsf{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} \downarrow_\kappa \leq ExtCoreR'_{\mathsf{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} \downarrow_\kappa$
$ExtCoreR'_{\mathsf{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} = (id, (\llbracket\Gamma\rrbracket, cmd, EHs_2 :: eh_{2\mathsf{ctx}}\llbracket\mathsf{skip}\rrbracket), \ell)$, $EHs \downarrow_\kappa \leq EHs_2 \downarrow_\kappa$
$eh_{2ctx}\llbracket\mathsf{skip}\rrbracket = (\cdot, eventType, x.cmd, \mathcal{E}' :: e_2, \llbracket\mathsf{skip}\rrbracket, id_e, \mathsf{blocking}, return)$, $\mathcal{E} \downarrow_\kappa \leq \mathcal{E}' \downarrow_\kappa$, $e_1 \downarrow_\kappa = e_2 \downarrow_\kappa$

(8) $labOf(e_1) \sqsubseteq \kappa$, $e_1 \downarrow_\kappa = e_1$, $e_1 = e_2$

By $labOf(e_2) = labOf(e_1) \neq \ell^-$, apply EVENTDEQUEUECOREBLOCKING1 to Σ_2, let $\Sigma_2 = \Sigma_{ctx2}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}$

(11) $eh'_{2\,ctx2}\llbracket\mathsf{skip}\rrbracket = (\cdot, eventType, x.cmd, \mathcal{E}', \llbracket\mathsf{skip}\rrbracket, id_e, \mathsf{blocking}, return)$

(12) $ExtCoreR_2{}'_{ctx2}(id) = ExtCoreR_{2\,ctx2}(id)[eh_{2\,ctx2}\llbracket\mathsf{skip}\rrbracket \Leftarrow eh'_{2\,ctx2}\llbracket\mathsf{skip}\rrbracket]$

(13) $ExtCoreRs_2{}'_{ctx2}(id) = ExtCoreRs_{2\,ctx2}(id)[ExtCoreR_{2\,ctx2}(id) \Leftarrow ExtCoreR'_{2\,ctx2}(id)]$

(14) $\Sigma_2{}'_{ctx2}(id) = \Sigma_{2\,ctx2}(id)[ExtCoreRs_{2\,ctx2}(id) \Leftarrow ExtCoreRs_2{}'_{ctx2}(id)]$

(15) $\Sigma_{2\,ctx2}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}; \mathcal{E}_2 \xrightarrow{\tau} \Sigma_2{}'_{ctx2}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}; \mathcal{E}'_2$, $\mathcal{E}'_2 = \mathcal{E}_2$

By (A5), (7), (11)

(16) $eh'_{ctx}\llbracket\mathsf{skip}\rrbracket \downarrow_\kappa \leq eh_2{}'_{ctx2}\llbracket\mathsf{skip}\rrbracket \downarrow_\kappa$

By (16), Lemma 6

(17) $\mathit{ExtCoreR}_{\mathbf{ctx}}[\![\Gamma, \mathsf{skip}]\!](id) \leq \mathit{ExtCoreR_2}'_{ctx2}[\![\Gamma,\ \mathsf{skip}]\!](id)$

(18) $\mathit{ExtCoreRs}'_{\mathbf{ctx}}[\![\Gamma, \mathsf{skip}]\!](id) \leq \mathit{ExtCoreRs_2}'_{ctx2}[\![\Gamma,\ \mathsf{skip}]\!](id)$

As $eh_{ctx}[\![\mathsf{skip}]\!]$ in $\mathit{ExtCoreR}_1$ and $eh_{2\,ctx2}[\![\mathsf{skip}]\!]$ in $\mathit{ExtCoreR}_2$ dequeue the same event, other eventhandlers are not changed

(19) $\mathit{ExtCoreRs}'_{\mathbf{ctx}}[\![\Gamma,\ \mathsf{skip}]\!](id) \leq_{c,e_1} \mathit{ExtCoreRs_2}'_{ctx2}[\![\Gamma,\ \mathsf{skip}]\!](id)$

(20) for all blocking handler e, $\mathit{ExtCoreRs}'_{\mathbf{ctx}}[\![\Gamma,\ \mathsf{skip}]\!](id) \leq_{c,e} \mathit{ExtCoreRs_2}'_{ctx2}[\![\Gamma,\ \mathsf{skip}]\!](id)$

By (19), (20)

(21) $\mathit{ExtCoreRs}'_{\mathbf{ctx}}[\![\Gamma,\ \mathsf{skip}]\!](id) \leq_{c} \mathit{ExtCoreRs_2}'_{ctx2}[\![\Gamma,\ \mathsf{skip}]\!](id)$

By (21), STATE

(22) $\Sigma'_{\mathbf{ctx}}(id) \leq \Sigma_2{}'_{ctx2}(id)$, that is $\Sigma'_1 \leq \Sigma'_2$

By $\mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$

(23) $\mathcal{E}'_1 \downarrow_\kappa \leq \mathcal{E}'_2 \downarrow_\kappa$

sub-subcase ii. $\ell^- \sqsubseteq \kappa$, $e_1 \not\sqsubseteq \kappa$

By EXTCORE1, EH

(1) $\mathit{ExtCoreR}_{\mathbf{ctx}}[\![\ \Gamma, \mathsf{skip}\ ]\!]_{id} \downarrow_\kappa = (id, ([\![\Gamma]\!], \mathit{cmd}, \mathit{EHs} \downarrow_\kappa :: eh_{\mathbf{ctx}}[\![\mathsf{skip}]\!] \downarrow_\kappa), \ell)$

(2) $eh_{ctx}[\![\mathsf{skip}]\!] \downarrow_\kappa = (\cdot, \mathit{eventType}, x.\mathit{cmd}, \mathcal{E} \downarrow_\kappa :: e_1 \downarrow_\kappa, [\![\mathsf{skip}]\!], id_e, \mathsf{blocking}, \mathit{return})$
$= (\cdot, \mathit{eventType}, x.\mathit{cmd}, \mathcal{E} \downarrow_\kappa, [\![\mathsf{skip}]\!], id_e, \mathsf{blocking}, \mathit{return}) = eh'_{ctx}[\![\mathsf{skip}]\!] \downarrow_\kappa$

By (A6), (A7), (A8), (A9)

(3) $\Sigma'_1 \downarrow_\kappa = \Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

By $\mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$, and (A10)

(4) $\mathcal{E}'_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$

sub-subcase iii. $\ell^- \not\sqsubseteq \kappa$

By EXTCORE2

(1) $\mathit{ExtCoreR} \downarrow_\kappa = \cdot$, $\mathit{ExtCoreR}_{\mathbf{ctx}}[\![\ \Gamma, \mathsf{skip}\ ]\!]_{id} \notin \Sigma_1 \downarrow_\kappa$

(2) $\mathit{ExtCoreR}' \downarrow_\kappa = \cdot$, $\mathit{ExtCoreR}_{\mathbf{ctx}}{}'[\![\ \Gamma, \mathsf{skip}\ ]\!]_{id} \notin \Sigma'_1 \downarrow_\kappa$

By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$, $\Sigma_1 = \Sigma_{ctx}[\![\ \Gamma, \mathsf{skip}\ ]\!]_{id}$, and $\Sigma'_1 = \Sigma'_{ctx}[\![\ \Gamma, \mathsf{skip}\ ]\!]_{id}$,
EVENTDEQUEUECOREBLOCKING1 only updates $\mathit{ExtCoreR}$, (1), (2)

(3) $\Sigma'_1 \downarrow_\kappa = \Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

By assumption

(4) $\mathcal{E}'_1 \downarrow_\kappa = \mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$

case: EVENTDEQUEUECOREBLOCKING2

$$\frac{\begin{array}{l}\Sigma_{ctx}(id) = (\Psi, \mathit{Tabs}, \mathit{ExtCoreRs}_{\mathbf{ctx}}(id), \cdots)\\ \mathit{ExtCoreR}_{\mathbf{ctx}}(id) \in \mathit{ExtCoreRs}_{\mathbf{ctx}}(id)\\ \mathit{ExtCoreR}_{\mathbf{ctx}}(id) = (id, ([\![]\!], \_, \mathit{EventHandlers} :: eh_{\mathbf{ctx}}), \ell)\\ eh_{\mathbf{ctx}} = (\cdot, \mathit{eventType}, x.\mathit{cmd}, e :: \mathcal{E}_1, [\![]\!], id_e, \mathsf{blocking}, \mathit{return})\\ e = (id'_e, \mathit{eventType}, \mathit{return}, \cdots, \kappa_e)\\ \kappa_e{=}\ell^- \qquad \mathit{fresh}(id_r) \qquad \psi = \mathsf{ProcBlkEvtState}(\kappa_e, id_r, e)\\ eh'_{\mathbf{ctx}} = (\cdot, \mathit{eventType}, x.\mathit{cmd}, \mathcal{E}_1, [\![]\!], id'_e, \mathsf{blocking}, \mathsf{some}(e, id_r))\\ \mathit{ExtCoreR}_{\mathbf{ctx}}{}'(id) = \mathit{ExtCoreR}_{\mathbf{ctx}}(id)[eh_{\mathbf{ctx}} \Leftarrow eh'_{\mathbf{ctx}}]\\ \mathit{ExtCoreRs}_{\mathbf{ctx}}{}'(id) = \mathit{ExtCoreRs}_{\mathbf{ctx}}(id)[\mathit{ExtCoreRs}_{\mathbf{ctx}}(id) \Leftarrow \mathit{ExtCoreRs}_{\mathbf{ctx}}{}'(id)]\\ \Sigma'_{ctx}(id) = \Sigma_{ctx}(id)[\mathit{ExtCoreRs}_{\mathbf{ctx}}(id) \Leftarrow \mathit{ExtCoreRs}'_{\mathbf{ctx}}(id)][\Psi \Leftarrow \Psi :: \psi]\end{array}}{\Sigma_{ctx}[\![\ \Gamma, \mathsf{skip}\ ]\!]_{id}; \mathcal{E} \longrightarrow \Sigma'_{ctx}[\![\ \Gamma, \mathit{cmd}[e/x]\ ]\!]_{id}; \mathcal{E}}\ \text{EVENTDEQUEUECOREBLOCKING2}$$

Let $\Sigma_1 = (\Psi_1, \mathit{Tabs}_1, \mathit{ExtCoreRs}_1, \mathit{progInjCSs}_1, \mathit{Exts}_1, \mathit{Cookies}_1, \mathit{MBookmarks}_1, \mathit{histories}_1, \mathit{UI}_1)$
$\Sigma_2 = (\Psi_2, \mathit{Tabs}_2, \mathit{ExtCoreRs}_2, \mathit{progInjCSs}_2, \mathit{Exts}_2, \mathit{Cookies}_2, \mathit{MBookmarks}_2, \mathit{histories}_2, \mathit{UI}_2)$

By assumptions

(A1) $\Sigma_1 = \Sigma_{ctx}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}$, $ExtCoreRs_1 = ExtCoreRs_1'' :: ExtCoreR$

(A2) $ExtCoreR = ExtCoreR_{\mathbf{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} = (id, (\llbracket \Gamma \rrbracket, cmd, EHs :: eh_{ctx}\llbracket \mathsf{skip} \rrbracket), \ell)$

(A3) $eh_{ctx}\llbracket \mathsf{skip} \rrbracket = (\cdot, eventType, x.cmd, \mathcal{E} :: e_1, \llbracket \mathsf{skip} \rrbracket, id_e, \mathsf{blocking}, return)$

(A4) $e_1 = (id_e', eventType, return, \cdots, \kappa_e)$, $\kappa_e = \ell^-$

(A5) $fresh(id_r)$

(A6) $eh'_{\mathbf{ctx}}\llbracket cmd[e_1/x] \rrbracket = (\cdot, eventType, x.cmd, \mathcal{E}, \llbracket cmd[e_1/x] \rrbracket, id_e{}', \mathsf{blocking}, \mathsf{some}(e_1, id_r))$

(A7) $ExtCoreR_{\mathbf{ctx}}{}'(id) = ExtCoreR_{\mathbf{ctx}}(id)[eh_{\mathbf{ctx}} \Leftarrow eh'_{\mathbf{ctx}}]$

(A8) $ExtCoreRs_1' = ExtCoreRs_{\mathbf{ctx}}{}'(id) = ExtCoreRs_{\mathbf{ctx}}(id)[ExtCoreRs_{\mathbf{ctx}}(id) \Leftarrow ExtCoreRs_{\mathbf{ctx}}{}'(id)]$

(A9) $\psi_1 = \mathsf{ProcBlkEvtState}(\kappa_e, id_r, e_1)$, $\Psi_1' = \Psi_1 :: \psi_1$

(A10) $\Sigma'_{ctx}(id) = \Sigma_{ctx}(id)[ExtCoreRs_{\mathbf{ctx}}(id) \Leftarrow ExtCoreRs'_{\mathbf{ctx}}(id)][\Psi_1 \Leftarrow \Psi_1']$

(A11) $\Sigma_1' = \Sigma'_{ctx}(id)$

(A12) $\Sigma_{ctx}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}; \mathcal{E}_1 \xrightarrow{updLab(e_1, \ell^-)} \Sigma'_{ctx}\llbracket \Gamma, cmd[e_1/x] \rrbracket_{id}; \mathcal{E}_1', \mathcal{E}_1' = \mathcal{E}_1$

sub-subcase i. $\ell^- \sqsubseteq \kappa$

By ExtCore1, EH

(1) $ExtCoreR_{\mathbf{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} \downarrow_\kappa = (id, (\llbracket \Gamma \rrbracket, cmd, EHs \downarrow_\kappa :: eh_{\mathbf{ctx}}\llbracket \mathsf{skip} \rrbracket \downarrow_\kappa), \ell)$

(2) $eh_{ctx}\llbracket \mathsf{skip} \rrbracket \downarrow_\kappa = (\cdot, eventType, x.cmd, \mathcal{E} \downarrow_\kappa :: e_1, \llbracket \mathsf{skip} \rrbracket, id_e, \mathsf{blocking}, return)$

By definition

(3) $ExtCoreR_{\mathbf{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} \downarrow_\kappa \in \Sigma_1 \downarrow_\kappa$

By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

(4) $ExtCoreRs_1 \downarrow_\kappa \leq_c ExtCoreRs_2 \downarrow_\kappa$

(5) $\Psi_1 \downarrow_\kappa \leq_c \Psi_2 \downarrow_\kappa$

By $ExtCoreRs_1 \downarrow_\kappa \leq_c ExtCoreRs_2 \downarrow_\kappa$

(6) $ExtCoreRs_1 \downarrow_\kappa \leq ExtCoreRs_2 \downarrow_\kappa$

(7) $\exists ExtCoreR'_{\mathbf{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} \in ExtCoreRs_2 \downarrow_\kappa$ s.t., $ExtCoreR_{\mathbf{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} \downarrow_\kappa \leq ExtCoreR'_{\mathbf{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} \downarrow_\kappa$
$ExtCoreR'_{\mathbf{ctx}}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id} = (id, (\llbracket \Gamma \rrbracket, cmd, EHs_2 :: eh_{2\mathbf{ctx}}\llbracket \mathsf{skip} \rrbracket), \ell)$, $EHs \downarrow_\kappa \leq EHs_2 \downarrow_\kappa$
$eh_{2ctx}\llbracket \mathsf{skip} \rrbracket = (\cdot, eventType, x.cmd, \mathcal{E}' :: e_2, \llbracket \mathsf{skip} \rrbracket, id_e, \mathsf{blocking}, return)$, $\mathcal{E} \downarrow_\kappa \leq \mathcal{E}' \downarrow_\kappa$, $e_1 \downarrow_\kappa = e_2 \downarrow_\kappa$

(8) $labOf(e_1) \sqsubseteq \kappa$, $e_1 \downarrow_\kappa = e_1$, $e_1 = e_2$

(9) $ExtCore_2 = ExtCoreR_{\mathbf{ctx}}{}'\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}$

(10) $ExtCoreRs_2 = ExtCoreRs_2'' :: ExtCoreR_2$

By $labOf(e_2) = \ell^-$, apply EventDequeueCoreBlocking2 to Σ_2, let $\Sigma_2 = \Sigma_{ctx2}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}$

(11) $fresh(id_r')$, $\psi_2 = \mathsf{ProcBlkEvtState}(\kappa_e, id_r', e_2)$, $\psi_2 = \psi_1$, $\Psi_2' = \Psi_2 :: \psi_2$

(12) $eh'_{2\,ctx2}\llbracket cmd[e_2/x] \rrbracket = (\cdot, eventType, x.cmd, \mathcal{E}', \llbracket cmd[e_2/x] \rrbracket, id_e{}', \mathsf{blocking}, \mathsf{some}(e_2, id_r'))$

(13) $ExtCoreR_2{}'_{ctx2}(id) = ExtCoreR_{2\,ctx2}(id)[eh_{2\,ctx2}\llbracket cmd[e_2/x] \rrbracket \Leftarrow eh'_{2\,ctx2}\llbracket cmd[e_2/x] \rrbracket]$

(14) $ExtCoreRs_2{}'_{ctx2}(id) = ExtCoreRs_{2\,ctx2}(id)[ExtCoreR_{2\,ctx2}(id) \Leftarrow ExtCoreR'_{2\,ctx2}(id)]$

(15) $\Sigma_2{}'_{ctx2}(id) = \Sigma_{2\,ctx2}(id)[ExtCoreRs_{2\,ctx2}(id) \Leftarrow ExtCoreRs_2{}'_{ctx2}(id)][\Psi_2 \Leftarrow \Psi_2']$

(16) $\Sigma_{2\,ctx2}\llbracket \Gamma, \mathsf{skip} \rrbracket_{id}; \mathcal{E}_2 \longrightarrow \Sigma_2{}'_{ctx2}\llbracket \Gamma, cmd[e_2/x] \rrbracket_{id}; \mathcal{E}_2', \mathcal{E}_2' = \mathcal{E}_2$

By (12)

(17) $eh'_{ctx}\llbracket \mathsf{skip} \rrbracket \downarrow_\kappa \leq eh_2{}'_{ctx2}\llbracket \mathsf{skip} \rrbracket \downarrow_\kappa$

By (17), Lemma 6

(18) $ExtCoreR'_{\mathbf{ctx}}(id) \leq ExtCoreR_2{}'_{ctx2}(id)$

(19) $ExtCoreRs'_{\mathbf{ctx}}(id) \leq ExtCoreRs_2{}'_{ctx2}(id)$

As both of $ExtCoreR_1$ and $ExtCoreR_2$ dequeue the same event without changing other parts

(20) $ExtCoreRs'_{\mathbf{ctx}}\llbracket \Gamma,\ \mathsf{skip} \rrbracket(id) \leq_c ExtCoreRs_2{}'_{ctx2}\llbracket \Gamma,\ \mathsf{skip} \rrbracket(id)$

By (5), $e_1 = e_2$

(21) $\Psi_1 \downarrow_\kappa = \psi :: \Psi_{11} :: \Psi_{12} :: \Psi_{13}$
$\psi \sim_b e_1$
Ψ_{11} contains states of the form $\mathsf{ProcBlkEvtState}(\_, \_, e_1)$

Ψ_{12} contains states of the form $\mathsf{DoneBlkEvtState}(\_, \_, e_1, \_)$
$\nexists \psi' \in \Psi_{13}$, s.t. $\psi' = \mathsf{ProcBlkEvtState}(\_, \_, e_1)$, or $\psi' = \mathsf{DoneBlkEvtState}(\_, \_, e_1, \_)$
(22) $\Psi_1' \downarrow_\kappa = \psi :: (\Psi_{11} :: \psi_1) :: \Psi_{12} :: \Psi_{13}$
(23) $\Psi_2 \downarrow_\kappa = \psi :: \Psi_{11} :: \Psi_{12} :: \Psi_{23}$
$\nexists \psi' \in \Psi_{23}$, s.t. $\psi' = \mathsf{ProcBlkEvtState}(\_, \_, e_1)$, or $\psi' = \mathsf{DoneBlkEvtState}(\_, \_, e_1, \_)$
(24) $\Psi_2' \downarrow_\kappa = \psi :: (\Psi_{11} :: \psi_2) :: \Psi_{12} :: \Psi_{23} = \psi :: (\Psi_{11} :: \psi_1) :: \Psi_{12} :: \Psi_{23}$
(25) $\Psi_1' \downarrow_\kappa \leq_{c,e} \Psi_2' \downarrow_\kappa$, $\Psi_1' \downarrow_\kappa \leq \Psi_2' \downarrow_\kappa$
By (25), and EVENTDEQUEUECOREBLOCKING2 does not change any other browserstate in Ψ_{23}
(26) $\Psi_1' \downarrow_\kappa \leq_c \Psi_2' \downarrow_\kappa$
By (20), (26), STATE
(27) $\Sigma'_{\mathit{ctx}}(id) \leq {\Sigma_2}'_{ctx2}(id)$, that is $\Sigma_1' \leq \Sigma_2'$
By $\mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$
(16) $\mathcal{E}_1' \downarrow_\kappa \leq \mathcal{E}_2' \downarrow_\kappa$
sub-subcase ii. $\ell^- \not\sqsubseteq \kappa$
By EXTCORE2
(1) $\mathit{ExtCoreR} \downarrow_\kappa = \cdot$, $\mathit{ExtCoreR}_{\mathit{ctx}} \llbracket \Gamma, \mathsf{skip} \rrbracket_{id} \notin \Sigma_1 \downarrow_\kappa$
(2) $\mathit{ExtCoreR}' \downarrow_\kappa = \cdot$, $\mathit{ExtCoreR}_{\mathit{ctx}}' \llbracket \Gamma, cmd[e_1/x] \rrbracket_{id} \notin \Sigma_1' \downarrow_\kappa$
(3) $\mathit{ExtCoreRs}_1 \downarrow_\kappa = \mathit{ExtCoreRs}_1' \downarrow_\kappa$
By BROWSERSTATE2
(4) $\psi_1 \downarrow_\kappa = \cdot$, $\Psi_1' \downarrow_\kappa = \Psi_1 \downarrow_\kappa$
EVENTDEQUEUECOREBLOCKING2 only updates $\mathit{ExtCoreR}$, adds ψ_1. By (3) and (4)
(5) $\Sigma_1' \downarrow_\kappa = \Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$
By assumption
(6) $\mathcal{E}_1' \downarrow_\kappa = \mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$

5.2 Proof of Lemma 15

The proofs for reading APIs and DOMNODEWRITE are straightforward as the structure of the tree does not change by these APIs. The equivalence relation can be established easily. The interesting APIs are the ones that change the tree structure.
case: DOMAPPENDCHILDREN(1A)

DOMAPPENDCHILDREN(A)

$$\frac{\begin{array}{l} \Sigma_{ctx}(id) = (\Psi, \mathit{Tabs} :: t_{\mathit{ctx}}(id), \cdots) \\ \ell_1 = \mathsf{ctxOfId}(\Sigma_{ctx} \llbracket \Gamma, \mathsf{let}\ x = \mathsf{domAPI.appendChild}(id_t, id_d, id_n, node_1, \ell_1)\ \mathsf{in}\ cmd \rrbracket_{id}, id) \\ \kappa = {\ell_1}^- \qquad t = (id_t, \_, \mathit{Docs} :: \mathit{Doc}, \cdots) \\ \mathit{Doc} = (id_d, \_, nodes, \_, \_) \qquad node \in_b nodes \qquad node = (id_n, (\cdots, \ell_{children}, \cdots), nodes_1, \cdots, \ell_n) \\ node_1 = (id_1, (\cdots, \ell_{parent}, \ell_{pre}, \cdot), \_, \ell_1) \qquad \nexists url \in attributes \\ \kappa \sqsubseteq {\ell_{children}}^* \qquad \kappa \sqsubseteq {\ell_{parent}}^* \qquad \kappa \sqsubseteq {\ell_{pre}}^* \qquad \ell_{succ} = \mathsf{nextSibleOf}(\mathsf{last}(nodes_1)) \qquad \kappa \sqsubseteq {\ell_{succ}}^* \\ \mathit{Doc}' = \mathit{Doc}[node \Leftarrow \mathsf{append}(node, node_1')] \qquad t' = t[\mathit{Doc} \Leftarrow \mathit{Doc}'] \qquad \Sigma'_{ctx}(id) = \Sigma_{ctx}(id)[t \Leftarrow t'] \end{array}}{\begin{array}{l} \Sigma_{ctx} \llbracket \Gamma, \mathsf{let}\ x = \mathsf{domAPI.appendChild}(id_t, id_d, id_n, node_1, \ell_1)\ \mathsf{in}\ cmd \rrbracket_{id} \\ \xrightarrow{\mathsf{call}(\mathsf{domAPI.appendChild}, (id_t, id_d, id_n, node_1, \ell_1), \mathsf{pointersOf}(node_1'), \kappa)} \Sigma'_{ctx} \llbracket \Gamma, cmd[\mathsf{pointersOf}(node_1')/x] \rrbracket_{id}; \cdot \end{array}}$$

Subcase: the script is a content script handler.

By assumptions
(A1) $\Sigma_1 = \Sigma_{ctx}(id) = (\Psi, \mathit{Tabs} :: t_{\mathit{ctx}}(id), \cdots)$
(A2) $\ell_1 = \mathsf{ctxOfId}(\Sigma_{ctx} \llbracket \Gamma, \mathsf{let}\ x = \mathsf{domAPI.appendChild}(id_t, id_d, id_n, node_a, \ell_1)\ \mathsf{in}\ cmd \rrbracket_{id}, id)$

(A3) $t_1 = (id_t, \_, Docs_1 :: Doc_1, \cdots)$
(A4) $Doc_1 = (id_d, \_, nodes_1 :: node, \_, \_)$
(A5) $node = (id_n, (\cdots, \ell_{children}, \cdots), nodes_1, \cdots, \ell_n)$
(A6) $node_1 = (id_1, (\cdots, \ell_{parent}, \ell_{pre}, \cdot), \_, \ell_1)$
(A7) $\nexists url \in attributes$
(A8) $\ell_{succ} = \mathsf{nextSibleOf}(\mathsf{last}(nodes_1))$
(A9) $\kappa \sqsubseteq {\ell_{children}}^*$, $\kappa \sqsubseteq {\ell_{parent}}^*$ $\kappa \sqsubseteq {\ell_{pre}}^*$ $\kappa \sqsubseteq {\ell_{succ}}^*$
(A10) $Doc_1' = Doc[node \Leftarrow \mathsf{append}(node, node_1')]$
(A11) $t_1' = t_1[Doc_1 \Leftarrow Doc_1']$
(A12) $\Sigma_1 = \Sigma'_{ctx}(id) = \Sigma_{ctx}(id)[t_1 \Leftarrow t_1']$

sub-subcase 1. The script's label ${\ell_1}^- \not\sqsubseteq \kappa_a$.
There are two cases here. One is that such append float some labels, the other is not.
In the latter case, the update does not affect the projection and less than relation.
In the former case, the projection of the dom will sever links between nodes.
In this case, the less than relation can be re-established using a different set of rules
that take into consideration the missing links.

sub-subcase 2. The script's label ${\ell_1}^- \sqsubseteq \kappa$.
Let $\Sigma_2 = \Sigma_{ctx2} \llbracket \Gamma, \mathsf{let}\ x = \mathsf{domAPI.appendChild}(id_t, id_d, id_n, node_1, \ell_1)\ \mathsf{in}\ cmd \rrbracket_{id}$
Here, no labels are tainted due to the addition of new node.
The resulting state still relation to each other using the same derivation.

Subcase: the script is a page script handler. The proofs are similar to above proofs.

case: DOMREMOVECHILD

$$\dfrac{\begin{array}{l}
\Sigma_{ctx}(id) = (\Psi, Tabs :: t_{\mathsf{ctx}}(id), \cdots) \\
\ell = \mathsf{ctxOfId}(\Sigma_{ctx} \llbracket \Gamma, \mathsf{let}\ x = \mathsf{domAPI.removeChild}(id_t, id_d, id_p, id_c)\ \mathsf{in}\ cmd \rrbracket_{id}, id) \\
\kappa = \ell^- \qquad t = (id_t, \_, Docs :: Doc, \cdots) \qquad Doc = (id_d, \_, nodes, \_, \_) \qquad node \in nodes \\
node = (id_p, m, nodes_1 :: node_1 :: node_c :: node_2 :: nodes_2, (\cdots, \ell_{children}, \cdots), \ell_p) \\
node_1 = (\cdots, (\cdots, \ell_{succ}, \cdots), \cdots) \qquad node_2 = (\cdots, (\cdots, \ell_{pre}, \cdots), \cdots) \\
\kappa \sqsubseteq {\ell_{children}}^* \qquad \kappa \sqsubseteq {\ell_{succ}}^* \qquad \kappa \sqsubseteq {\ell_{pre}}^* \\
node_1' = node_1[\ell_{succ} \Leftarrow \kappa \rhd_{tnt} \ell_{succ}] \qquad node_2' = node_1[\ell_{pre} \Leftarrow \kappa \rhd_{tnt} \ell_{pre}] \\
node' = (id_p, m, nodes_1 :: node_1' :: node_2' :: nodes_2, (\cdots, \kappa \rhd_{tnt} \ell_{children}, \cdots), \ell_p) \\
Doc' = \mathsf{updateQ}(Doc[node \Leftarrow node'], \kappa, node_c) \\
t' = t[Doc \Leftarrow Doc'] \qquad \Sigma_{ctx}(id)' = \Sigma_{ctx}(id)[t \Leftarrow t']
\end{array}}{\begin{array}{l}
\Sigma_{ctx} \llbracket \Gamma, \mathsf{let}\ x = \mathsf{domAPI.removeChild}(id_t, id_d, id_p, id_c)\ \mathsf{in}\ cmd \rrbracket_{id} \\
\xrightarrow{\mathsf{call}(\mathsf{domAPI.removeNode}, (id_t, id_d, id_p, id_c), \mathsf{pointersOf}(node), \kappa)} \Sigma'_{ctx} \llbracket \Gamma, cmd[\mathsf{pointersOf}(node)/x] \rrbracket_{id}; \cdot
\end{array}}\ \textsc{DomRemoveChild}$$

Similar to the previous case, we have two subcases: one is the script's label is lower than or equal to κ_a, the other is not.

The first case is straightforward. The equivalent state also takes the same step and the projected tree structures stay the same.

For the second case, we need to show the state after removal of the child is still equivalent to Σ_2.

There are two cases here. One is that such append float some labels, the other is not. In the latter case, the update does not affect the projection and less than relation. In the former case, the projection of the dom will sever links between nodes. In this case, the less than relation can be re-established using a different set of rules that take case of the missing links and extra node in Σ_2.

5.3 Proof of Lemma 14

case: BOOKMARKGET

$$\dfrac{\begin{array}{l}\Sigma = (\Psi, \cdots, \mathit{MBookmarks}, \cdots) \qquad \Psi = \Psi' :: \mathsf{chrome.bookmarks.get}(\kappa, id_{cb}, ids) \\ \mathit{bookmarks} = \mathsf{selectBookmark}(\mathit{MBookmarks}, \kappa) \\ \mathit{result} = \mathsf{bookmarkQuery}(\mathit{bookmarks}, \mathsf{chrome.bookmarks.get}, ids) \\ \mathit{fresh}(id_e) \qquad e = (id_e, id_{cb}, \mathsf{none}, \mathit{result}, \kappa) \qquad \Sigma' = \Sigma[\Psi \Leftarrow \Psi']\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{chrome.bookmarks.get}(\kappa, id_{cb}, ids)) = \Sigma'; e}\ \textsc{bookmarkGet}$$

Let $\Sigma_1 = (\Psi_1, \mathit{Tabs}_1, \mathit{ExtCoreRs}_1, \mathit{progInjCSs}_1, \mathit{Exts}_1, \mathit{Cookies}_1, \mathit{MBookmarks}_1, \mathit{histories}_1, \mathit{UI}_1)$
$\Sigma_2 = (\Psi_2, \mathit{Tabs}_2, \mathit{ExtCoreRs}_2, \mathit{progInjCSs}_2, \mathit{Exts}_2, \mathit{Cookies}_2, \mathit{MBookmarks}_2, \mathit{histories}_2, \mathit{UI}_2)$

By assumptions

(A1) $\Psi_1 = \Psi_1' :: \psi_1$
(A2) $\psi_1 = \mathsf{chrome.bookmarks.get}(\kappa_1, id_{cb}, ids)$
(A3) $\mathit{bookmarks}_1 = \mathsf{selectBookmark}(\mathit{MBookmarks}_1, \kappa)$
(A4) $\mathit{result}_1 = \mathsf{bookmarkQuery}(\mathit{bookmarks}_1, \mathsf{chrome.bookmarks.get}, ids)$
(A5) $\mathit{fresh}(id_{e1})$
(A6) $e_1 = (id_{e1}, id_{cb}, \mathsf{none}, \mathit{result}_1, \kappa)$
(A7) $\Sigma_1' = \Sigma_1[\Psi_1 \Leftarrow \Psi_1']$

sub-subcase i. $\kappa_1 \sqsubseteq \kappa$

By BROWSERSTATE1

(1) $\psi_1 \downarrow_\kappa = \psi_1$

By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$, Lemma 11, and (1)

(2) $\Psi_1 \downarrow_\kappa \leq_c \Psi_2 \downarrow_\kappa$
(3) $\exists \psi_2 \in \Psi_2$ s.t. $\psi_1 \downarrow_\kappa \leq \psi_2 \downarrow_\kappa$, that is $\psi_2 = \psi_1 = \mathsf{chrome.bookmarks.get}(\kappa_1, id_{cb}, ids)$,
$\Psi_2 = \Psi_2' :: \psi_2$, $\Psi_1' \downarrow_\kappa \leq_c \Psi_2' \downarrow_\kappa$

Apply BOOKMARKGET to Σ_2

(4) $\mathit{bookmarks}_2 = \mathsf{selectBookmark}(\mathit{MBookmarks}_2, \kappa_1)$
(5) $\mathit{result}_2 = \mathsf{bookmarkQuery}(\mathit{bookmarks}_2, \mathsf{chrome.bookmarks.get}, ids)$
(6) $\mathit{fresh}(id_{e2})$
(7) $e_2 = (id_{e2}, id_{cb}, \mathsf{none}, \mathit{result}_2, \kappa)$
(8) $\Sigma_2' = \Sigma_2[\Psi_2 \Leftarrow \Psi_2']$

By $\mathit{MBookmarks}_1 \downarrow_\kappa \leq_{bkm} \mathit{MBookmarks}_2 \downarrow_\kappa$, $\kappa_1 \sqsubseteq \kappa$, and the definition of selectBookmark

(9) $\mathit{bookmarks}_2 = \mathit{bookmarks}_1$
(10) $\mathit{result}_2 = \mathit{result}_1$
(11) $e_2 = e_1$

By EVENT1, (11)

(12) $e_1 \downarrow_\kappa \leq e_2 \downarrow_\kappa$

By STATE, (A7), (8), (3)

(13) $\Sigma_1' \downarrow_\kappa \leq \Sigma_2' \downarrow_\kappa$

sub-subcase ii. $\kappa_1 \not\sqsubseteq \kappa$

By BROWSERSTATE2, Lemma 12

(1) $\psi_1 \downarrow_\kappa = \cdot$
(2) $\Psi_1' \downarrow_\kappa \leq_c \Psi_1 \downarrow_\kappa$

By (A6), EVENT2

(3) $e_1 \downarrow_\kappa = \cdot$

By STATE, (A7), (2), $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

(4) $\Sigma_1' \downarrow_\kappa \leq \Sigma_1 \downarrow_\kappa$

(5) $\Sigma_1' \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

The proofs for BOOKMARKGETCHILDREN, BOOKMARKGETRECENT, BOOKMARKGETTREE, BOOKMARKGETSUBTREE, BOOKMARKSEARCH are similar to the proofs for this rule.

case: BOOKMARKCREATE(1)

$$\dfrac{\begin{array}{l}\Sigma = (\Psi, \cdots, \mathit{MBookmarks}, \cdots)\\ \Psi = \Psi' :: \mathsf{chrome.bookmarks.create}(\kappa, id_{cb}, (id_{parent}, \mathit{positionIndex}, \mathit{title}, \mathit{url}))\\ \mathit{MBookmarks} = \mathit{MBookmarks}'' :: (\mathit{bookmarks}, \kappa)\\ (\mathit{bookmarks}', \mathit{result}) = \mathsf{bookmarkWrite}(\mathit{bookmarks}, \mathsf{chrome.bookmarks.create},\\ \qquad\qquad (id_{parent}, \mathit{positionIndex}, \mathit{title}, \mathit{url}))\\ \mathit{fresh}(id_e)\\ e = (id_e, id_{cb}, \mathsf{none}, \mathit{result}, \kappa) \qquad \mathit{MBookmarks}' = \mathit{MBookmarks}'' :: (\mathit{bookmark}', \kappa)\\ \Sigma' = \Sigma[\Psi \Leftarrow \Psi'][\mathit{MBookmarks} \Leftarrow \mathit{MBookmarks}']\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{chrome.bookmarks.create}(\kappa, id_{cb}, (id_{parent}, \mathit{positionIndex}, \mathit{title}, \mathit{url}))) = \Sigma'; e}\ \text{BOOKMARKCREATE(1)}$$

Let $\Sigma_1 = (\Psi_1, \mathit{Tabs}_1, \mathit{ExtCoreRs}_1, \mathit{progInjCSs}_1, \mathit{Exts}_1, \mathit{Cookies}_1, \mathit{MBookmarks}_1, \mathit{histories}_1, \mathit{UI}_1)$
$\Sigma_2 = (\Psi_2, \mathit{Tabs}_2, \mathit{ExtCoreRs}_2, \mathit{progInjCSs}_2, \mathit{Exts}_2, \mathit{Cookies}_2, \mathit{MBookmarks}_2, \mathit{histories}_2, \mathit{UI}_2)$

By assumptions

(A1) $\Psi_1 = \Psi_1' :: \psi_1$

(A2) $\psi_1 = \mathsf{chrome.bookmarks.create}(\kappa_1, id_{cb}, (id_{parent}, \mathit{positionIndex}, \mathit{title}, \mathit{url}))$

(A3) $\mathit{MBookmarks}_1 = \mathit{MBookmarks}_1'' :: \mathit{MBookmark}_1$

(A4) $\mathit{MBookmark}_1 = (\mathit{bookmarks}_1, \kappa_1)$

(A5) $(\mathit{bookmarks}_1', \mathit{result}_1) = \mathsf{bookmarkWrite}(\mathit{bookmarks}_1, \mathsf{chrome.bookmarks.create},$

(A6) $\qquad (id_{parent}, \mathit{positionIndex}, \mathit{title}, \mathit{url}))$

(A7) $\mathit{fresh}(id_{e1})$

(A8) $e_1 = (id_{e1}, id_{cb}, \mathsf{none}, \mathit{result}_1, \kappa_1)$

(A9) $\mathit{MBookmarks}_1' = \mathit{MBookmarks}_1[\mathit{bookmark}_1 \Leftarrow \mathit{bookmark}_1']$

(A10) $\mathit{MBookmarks}_1' = \mathit{MBookmarks}_1[\mathit{MBookmark}_1 \Leftarrow \mathit{MBookmark}_1']$

(A11) $\Sigma_1' = \Sigma_1[\Psi_1 \Leftarrow \Psi_1'][\mathit{MBookmarks}_1 \Leftarrow \mathit{MBookmarks}_1']$

sub-subcase i. $\kappa_1 \sqsubseteq \kappa$

By BROWSERSTATE1

(1) $\psi_1 \downarrow_\kappa = \psi_1$

By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$, Lemma 11, and (1)

(2) $\Psi_1 \downarrow_\kappa \leq_c \Psi_2 \downarrow_\kappa$

(3) $\exists \psi_2 \in \Psi_2$ s.t. $\psi_1 \downarrow_\kappa \leq \psi_2 \downarrow_\kappa$,
that is $\psi_2 = \psi_1 = \mathsf{chrome.bookmarks.create}(\kappa_1, id_{cb}, (id_{parent}, \mathit{positionIndex}, \mathit{title}, \mathit{url}))$,
$\Psi_2 = \Psi_2' :: \psi_2$, $\Psi_1' \downarrow_\kappa \leq_c \Psi_2' \downarrow_\kappa$

By MBOOKMARK1

(4) $\mathit{MBookmark}_1 \downarrow_\kappa = \mathit{MBookmark}_1$

By $\mathit{MBookmarks}_1 \downarrow_\kappa \leq_{bkm} \mathit{MBookmarks}_2 \downarrow_\kappa$, (4), MBOOKMARKS

(5) $\exists \mathit{MBookmark}_2 \in \mathit{MBookmarks}_2$, s.t. $\mathit{MBookmark}_1 \downarrow_\kappa \leq \mathit{MBookmark}_2 \downarrow_\kappa$,
that is $\mathit{MBookmark}_2 = \mathit{MBookmark}_1$

(6) $\mathit{MBookmarks}_2 = \mathit{MBookmarks}_2'' :: \mathit{MBookmark}_2$

(7) $\mathit{MBookmarks}_2'' \downarrow_\kappa = \mathit{MBookmarks}_1'' \downarrow_\kappa$

By (3), (5), apply BOOKMARKCREATE(1) to Σ_2

(8) $\mathit{MBookmark}_2 = (\mathit{bookmarks}_1, \kappa_1)$

(9) $(\mathit{bookmarks}_1', \mathit{result}_1) = \mathsf{bookmarkWrite}(\mathit{bookmarks}_1, \mathsf{chrome.bookmarks.create},$

$(id_{parent}, positionIndex, title, url))$

(10) $fresh(id_{e2})$

(11) $e_2 = (id_{e2}, id_{cb}, \mathsf{none}, result_1, \kappa_1)$

(12) $MBookmarks'_2 = MBookmarks_2[bookmark_1 \Leftarrow bookmark'_1]$

(13) $MBookmarks'_2 = MBookmarks_2[MBookmark_2 \Leftarrow MBookmark'_2]$

(14) $\Sigma'_2 = \Sigma_2[\Psi_2 \Leftarrow \Psi'_2][MBookmarks_2 \Leftarrow MBookmarks'_2]$

By $MBookmarks_1 \downarrow_\kappa \leq_{bkm} MBookmarks_2 \downarrow_\kappa$, $\kappa_1 \sqsubseteq \kappa$, and the definition of selectBookmark

(15) $bookmarks_2 = bookmarks_1$

(16) $MBookmarks'_2 = MBookmarks'_1$

By (A10), (7), (13), (16), BOOKMARKS

(17) $MBookmarks'_1 \leq_{bkm} MBookmarks'_2$

By EVENT1, (A8)

(18) $e_1 \downarrow_\kappa \leq e_2 \downarrow_\kappa$

By STATE, (A11), (3), (14), (17)

(19) $\Sigma'_1 \downarrow_\kappa \leq \Sigma'_2 \downarrow_\kappa$

sub-subcase ii. $\kappa_1 \not\sqsubseteq \kappa$

By BROWSERSTATE2, Lemma 12

(1) $\psi_1 \downarrow_\kappa = \cdot$

(2) $\Psi'_1 \downarrow_\kappa \leq_c \Psi_1 \downarrow_\kappa$

By MBOOKMARK2

(3) $MBookmark_1 \downarrow_\kappa = MBookmark'_1 \downarrow_\kappa = \cdot$

By (A3), (A10), (3)

(4) $MBookmarks_1 \downarrow_\kappa = MBookmarks'_1 \downarrow_\kappa$

By STATE, (A11), (2), (4), $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

(5) $\Sigma'_1 \downarrow_\kappa \leq \Sigma_1 \downarrow_\kappa$

(6) $\Sigma'_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

By (A8), EVENT2

(7) $e_1 \downarrow_\kappa = \cdot$

The proofs for BOOKMARKMOVE(1), BOOKMARKUPDATE(1), BOOKMARKREMOVE(1), BOOKMARKREMOVETREE(1) are similar to the proofs for the rule BOOKMARKCREATE(1).

case: BOOKMARKCREATE(2)

$$\frac{\begin{array}{l}\Sigma = (\Psi, \cdots, MBookmarks, \cdots) \\ \Psi = \Psi' :: \mathsf{chrome.bookmarks.create}(\kappa, id_{cb}, (id_{parent}, positionIndex, title, url)) \\ \nexists MBookmark \in MBookmarks \text{ s.t. } labOf(MBookmark) = \kappa \\ bookmarks_1 = \mathsf{selectBookmark}(MBookmarks, \kappa) \\ (bookmarks'_1, result) = \mathsf{bookmarkWrite}(bookmarks_1, \mathsf{chrome.bookmarks.create}, \\ \qquad (id_{parent}, positionIndex, title, url)) \\ fresh(id_e) \\ e = (id_e, id_{cb}, \mathsf{none}, result, \kappa) \qquad MBookmarks' = MBookmarks :: (bookmarks'_1, \kappa) \\ \Sigma' = \Sigma[\Psi \Leftarrow \Psi'][MBookmarks \Leftarrow MBookmarks']\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{chrome.bookmarks.create}(\kappa, id_{cb}, (id_{parent}, positionIndex, title, url))) = \Sigma'; e} \text{ BOOKMARKCREATE(2)}$$

Let $\Sigma_1 = (\Psi_1, Tabs_1, ExtCoreRs_1, progInjCSs_1, Exts_1, Cookies_1, MBookmarks_1, histories_1, UI_1)$
$\Sigma_2 = (\Psi_2, Tabs_2, ExtCoreRs_2, progInjCSs_2, Exts_2, Cookies_2, MBookmarks_2, histories_2, UI_2)$

By assumptions

(A1) $\Psi_1 = \Psi'_1 :: \psi_1$

(A2) $\psi_1 = \textsf{chrome.bookmarks.create}(\kappa_1, id_{cb}, (id_{parent}, positionIndex, title, url))$
(A3) $\nexists MBookmark \in MBookmarks_1$ s.t. $labOf(MBookmark_1) = \kappa_1$
(A4) $bookmarks_1 = \textsf{selectBookmark}(MBookmarks, \kappa_1)$
(A5) $(bookmarks'_1, result_1) = \textsf{bookmarkWrite}(bookmarks_1, \textsf{chrome.bookmarks.create},$
(A6) $(id_{parent}, positionIndex, title, url))$
(A7) $MBookmark_1 = (bookmarks'_1, \kappa_1)$
(A8) $MBookmarks'_1 = MBookmarks_1 :: MBookmark_1$
(A9) $fresh(id_{e1})$
(A10) $e_1 = (id_{e1}, id_{cb}, \textsf{none}, result_1, \kappa_1)$
(A11) $\Sigma'_1 = \Sigma_1[\Psi_1 \Leftarrow \Psi'_1][MBookmarks_1 \Leftarrow MBookmarks'_1]$

sub-subcase i. $\kappa_1 \sqsubseteq \kappa$

By BROWSERSTATE1
(1) $\psi_1 \downarrow_\kappa = \psi_1$

By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$, Lemma 11, and (1)
(2) $\Psi_1 \downarrow_\kappa \leq_c \Psi_2 \downarrow_\kappa$
(3) $\exists \psi_2 \in \Psi_2$ s.t. $\psi_1 \downarrow_\kappa \leq \psi_2 \downarrow_\kappa$,
that is $\psi_2 = \psi_1 = \textsf{chrome.bookmarks.create}(\kappa_1, id_{cb}, (id_{parent}, positionIndex, title, url))$,
$\Psi_2 = \Psi'_2 :: \psi_2$, $\Psi'_1 \downarrow_\kappa \leq_c \Psi'_2 \downarrow_\kappa$

By MBOOKMARK1
(4) $\forall MBookmark \in MBookmarks_1$,
if $labOf(MBookmark) = \kappa_1$, $MBookmark \downarrow_\kappa = MBookmark$
$MBookmark \in MBookmarks_1 \downarrow_\kappa$
(5) $\forall MBookmark \in MBookmarks_2$,
if $labOf(MBookmark) = \kappa_1$, $MBookmark \downarrow_\kappa = MBookmark$
$MBookmark \in MBookmarks_2 \downarrow_\kappa$

By (A3), (4)
(6) $\nexists MBookmark \in MBookmarks_1 \downarrow_\kappa$ s.t. $labOf(MBookmark) = \kappa_1$

By $MBookmarks_1 \downarrow_\kappa \leq_{bkm} MBookmarks_2 \downarrow_\kappa$, MBOOKMARKS, (5), (6)
(7) $MBookmarks_2 \downarrow_\kappa = MBookmarks_1 \downarrow_\kappa$
(8) $\nexists MBookmark \in MBookmarks_2$ s.t. $labOf(MBookmark) = \kappa_1$
Proof: if $\nexists MBookmark \in MBookmarks_2$ s.t. $labOf(MBookmark) = \kappa_1$
then $MBookmark \in MBookmarks_2 \downarrow_\kappa$ which conflicts with (7)

By (3), (8), apply BOOKMARKCREATE(2) to Σ_2
(9) $bookmarks_2 = \textsf{selectBookmark}(MBookmarks_2, \kappa_1)$
(10) $(bookmarks'_2, result_2) = \textsf{bookmarkWrite}(bookmarks_2, \textsf{chrome.bookmarks.create},$
(11) $(id_{parent}, positionIndex, title, url))$
(12) $MBookmark_2 = (bookmarks'_2, \kappa_1)$
(13) $MBookmarks'_2 = MBookmarks_2 :: MBookmark_2$
(14) $fresh(id_{e2})$
(15) $e_2 = (id_{e2}, id_{cb}, \textsf{none}, result_2, \kappa_1)$
(16) $\Sigma'_2 = \Sigma_2[\Psi_2 \Leftarrow \Psi'_2][MBookmarks_2 \Leftarrow MBookmarks'_2]$

By $MBookmarks_1 \downarrow_\kappa \leq_{bkm} MBookmarks_2 \downarrow_\kappa$, $\kappa_1 \sqsubseteq \kappa$, and the definition of selectBookmark, (4), (5)
(17) $bookmarks_2 = bookmarks_1$

By (A5), (10), (17)
(18) $bookmarks'_2 = bookmarks'_1$
(19) $result_2 = result_1$

By (17), (A7), (12)
(20) $MBookmark_2 = MBookmark_2$

By (A8), (7), (13), (20), BOOKMARKS

(21) $\mathit{MBookmarks}'_1 = \mathit{MBookmarks}'_2$

(22) $\mathit{MBookmarks}'_1 \leq_{bkm} \mathit{MBookmarks}'_2$

By STATE, (A11), (3), (16), (22)

(23) $\Sigma'_1 \downarrow_\kappa \leq \Sigma'_2 \downarrow_\kappa$

By EVENT1, (A8), (15), (19)

(24) $e_1 \downarrow_\kappa \leq e_2 \downarrow_\kappa$

sub-subcase ii. $\kappa_1 \not\sqsubseteq \kappa$

By BROWSERSTATE2, Lemma 12

(1) $\psi_1 \downarrow_\kappa = \cdot$

(2) $\Psi'_1 \downarrow_\kappa \leq_c \Psi_1 \downarrow_\kappa$

By MBOOKMARK2, (A7)

(3) $\mathit{MBookmark}_1 \downarrow_\kappa = \cdot$

By (A8), (3)

(4) $\mathit{MBookmarks}_1 \downarrow_\kappa = \mathit{MBookmarks}'_1 \downarrow_\kappa$

(5) $\mathit{MBookmarks}'_1 \downarrow_\kappa \leq_{bkm} \mathit{MBookmarks}_1 \downarrow_\kappa$

By STATE, (A11), (2), (5), $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

(5) $\Sigma'_1 \downarrow_\kappa \leq \Sigma_1 \downarrow_\kappa$

(6) $\Sigma'_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

By (A10), EVENT2

(7) $e_1 \downarrow_\kappa = \cdot$

The proofs for BOOKMARKMOVE(2), BOOKMARKUPDATE(2), BOOKMARKREMOVE(2), BOOKMARKREMOVE-TREE(2) are similar to the rule BOOKMARKCREATE(2).

case: COOKIEWEBSITESET(1)

$$\frac{\begin{array}{l}\Sigma = (\Psi, \cdots, \mathit{Cookies}, \cdots) \\ \Psi = \Psi' :: \mathsf{website.cookieSet}(\kappa, id_{cb}, (\mathit{name}, \mathit{value}, \mathit{url})) \qquad \mathit{Cookie} \in \mathit{Cookies} \\ \mathit{Cookie} = (\mathit{name}, \mathit{value}_1, \mathit{url}, \kappa) \qquad \mathit{Cookie}' = \mathit{Cookie}[\mathit{value}_1 \Leftarrow \mathit{value}] \\ \mathit{Cookies}' = \mathit{Cookies}[\mathit{Cookie} \Leftarrow \mathit{Cookie}'] \qquad \Sigma' = \Sigma[\Psi \Leftarrow \Psi'][\mathit{Cookies} \Leftarrow \mathit{Cookies}'] \\ \mathit{fresh}(id_e) \qquad e = (id_e, id_{cb}, \mathsf{none}, \mathit{info}(\mathit{Cookie}'), \kappa)\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{website.cookieSet}(\kappa, id_{cb}, (\mathit{name}, \mathit{value}, \mathit{url}))) = \Sigma'; e} \ \text{COOKIEWEBSITESET(1)}$$

Let $\Sigma_1 = (\Psi_1, \mathit{Tabs}_1, \mathit{ExtCoreRs}_1, \mathit{progInjCSs}_1, \mathit{Exts}_1, \mathit{Cookies}_1, \mathit{MBookmarks}_1, \mathit{histories}_1, \mathit{UI}_1)$
$\Sigma_2 = (\Psi_2, \mathit{Tabs}_2, \mathit{ExtCoreRs}_2, \mathit{progInjCSs}_2, \mathit{Exts}_2, \mathit{Cookies}_2, \mathit{MBookmarks}_2, \mathit{histories}_2, \mathit{UI}_2)$

By assumptions

(A1) $\Psi_1 = \Psi'_1 :: \psi_1$

(A2) $\psi_1 = \mathsf{website.cookieSet}(\kappa_1, id_{cb}, (\mathit{name}, \mathit{value}, \mathit{url}))$

(A3) $\mathit{Cookie}_1 \in \mathit{Cookies}_1$

(A4) $\mathit{Cookie}_1 = (\mathit{name}, \mathit{value}_1, \mathit{url}, \kappa_1)$

(A5) $\mathit{Cookie}'_1 = \mathit{Cookie}_1[\mathit{value}_1 \Leftarrow \mathit{value}]$

(A6) $\mathit{Cookies}'_1 = \mathit{Cookies}_1[\mathit{Cookie}_1 \Leftarrow \mathit{Cookie}'_1]$

(A7) $\mathit{fresh}(id_{e1})$

(A8) $e_1 = (id_{e1}, id_{cb}, \mathsf{none}, \mathit{info}(\mathit{Cookie}'_1), \kappa)$

(A9) $\Sigma'_1 = \Sigma_1[\Psi_1 \Leftarrow \Psi'_1][\mathit{Cookies}_1 \Leftarrow \mathit{Cookies}'_1]$

sub-subcase i. $\kappa_1 \sqsubseteq \kappa$

By BROWSERSTATE1

(1) $\psi_1 \downarrow_\kappa = \psi_1$

By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$, Lemma 11, and (1)

(2) $\Psi_1 \downarrow_\kappa \leq_c \Psi_2 \downarrow_\kappa$

(3) $\exists \psi_2 \in \Psi_2$ s.t. $\psi_1 \downarrow_\kappa \leq \psi_2 \downarrow_\kappa$,
that is $\psi_2 = \psi_1 = \mathsf{website.cookieSet}(\kappa_1, id_{cb}, (name, value, url))$,
$\Psi_2 = \Psi_2' :: \psi_2$, $\Psi_1' \downarrow_\kappa \leq_c \Psi_2' \downarrow_\kappa$

By COOKIE1

(4) $Cookie_1 \downarrow_\kappa = Cookie_1$

By $Cookies_1 \downarrow_\kappa \leq cookies_2 \downarrow_\kappa$, (4)

(5) $\exists Cookie_2 \in Cookies_2$ s.t. $Cookie_1 \downarrow_\kappa \leq Cookie_2 \downarrow_\kappa$,
that is $Cookie_2 = Cookie_1 = (name, value_1, url, \kappa_1)$

By (3), (5), apply COOKIEWEBSITESET(1) to Σ_2

(6) $Cookie_2' = Cookie_2[value_1 \Leftarrow value]$

(7) $Cookies_2' = Cookies_2[Cookie_2 \Leftarrow Cookie_2']$

(8) $fresh(id_{e2})$

(9) $e_2 = (id_{e2}, id_{cb}, \mathsf{none}, info(Cookie_2'), \kappa_1)$

(10) $\Sigma_2' = \Sigma_2[\Psi_1 \Leftarrow \Psi_1'][Cookies_1 \Leftarrow Cookies_1']$

By (5), (6)

(11) $Cookie_2' = Cookie_1'$

By (A6), (7), Lemma 6

(12) $Cookies_2' = Cookies_1'$

By STATE, (A9), (3), (10), (12)

(13) $\Sigma_1' \downarrow_\kappa \leq \Sigma_2' \downarrow_\kappa$

By EVENT1, (A8), (9), (11)

(14) $e_1 \downarrow_\kappa \leq e_2 \downarrow_\kappa$

sub-subcase ii. $\kappa_1 \not\sqsubseteq \kappa$

By BROWSERSTATE2, Lemma 12

(1) $\psi_1 \downarrow_\kappa = \cdot$

(2) $\Psi_1' \downarrow_\kappa \leq_c \Psi_1 \downarrow_\kappa$

By COOKIE2, (A4), (A5)

(3) $Cookie_1 \downarrow_\kappa = Cookie_1' \downarrow_\kappa \cdot$

By (A6), Lemma 6

(4) $Cookies_1' = Cookies_1$

By STATE, (A9), (3), (4)

(5) $\Sigma_1' \downarrow_\kappa \leq \Sigma_1 \downarrow_\kappa$

By (A8) , EVENT2

(6) $e_1 \downarrow_\kappa = \cdot$

The proofs for COOKIEWEBSITESET(2), COOKIESCRIPTSET(1) are similar to the proofs for COOKIEWEBSITESET(1).

case: COOKIECOREGETALL

Let $\Sigma_1 = (\Psi_1, Tabs_1, ExtCoreRs_1, progInjCSs_1, Exts_1, Cookies_1, MBookmarks_1, histories_1, UI_1)$
$\Sigma_2 = (\Psi_2, Tabs_2, ExtCoreRs_2, progInjCSs_2, Exts_2, Cookies_2, MBookmarks_2, histories_2, UI_2)$

By assumptions

(A1) $\Psi_1 = \Psi_1' :: \psi_1$

(A2) $\psi_1 = \mathsf{chrome.cookies.getAll}(\kappa_1, id_{cb}, filter)$

(A3) $\forall Cookie \in Cookies_1, Cookie = (\_, \_, \_, \kappa_c)$
    if $\mathsf{matchFilter}(Cookie, filter) = \mathsf{true} \wedge \kappa_c \sqsubseteq \kappa_1$
        Add $Cookie$ to $Cookies_{results_1}$

(A4) $\mathit{fresh}(id_{e1})$
$e_1 = (id_{e1}, id_{cb}, \mathsf{none}, \mathit{Cookies}_{\mathit{results}_1}, \kappa_1)$
(A5) $\Sigma_1' = \Sigma_1[\Psi_1 \Leftarrow \Psi_1']$

sub-subcase i. $\kappa_1 \sqsubseteq \kappa$

By BROWSERSTATE1
(1) $\psi_1 \downarrow_\kappa = \psi_1$
By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$, Lemma 11, and (1)
(2) $\Psi_1 \downarrow_\kappa \leq_c \Psi_2 \downarrow_\kappa$
(3) $\exists \psi_2 \in \Psi_2$ s.t. $\psi_1 \downarrow_\kappa \leq \psi_2 \downarrow_\kappa$,
that is $\psi_2 = \psi_1 = \mathsf{chrome.cookies.getAll}(\kappa_1, id_{cb}, \mathit{filter})$,
$\Psi_2 = \Psi_2' :: \psi_2$, $\Psi_1' \downarrow_\kappa \leq_c \Psi_2' \downarrow_\kappa$
By COOKIE1, $\mathit{Cookies}_1 \downarrow_\kappa \leq \mathit{Cookies}_2 \downarrow_\kappa$
(4) $\forall \mathit{Cookie} \in \mathit{Cookie}_{\mathit{result}_1}$, $\mathit{Cookie} \downarrow_\kappa = \mathit{Cookie}$
$\exists \mathit{Cookie}' \in \mathit{Cookies}_2$ s.t. $\mathit{Cookie} \downarrow_\kappa \leq \mathit{Cookie}' \downarrow_\kappa$,
that is $\mathit{Cookie}' = \mathit{Cookie}$
By (3), apply COOKIECOREGETALL(2) to Σ_2
(5) $\forall \mathit{Cookie} \in \mathit{Cookies}_2$, $\mathit{Cookie} = (\_, \_, \_, \kappa_c)$
if $\mathsf{matchFilter}(\mathit{Cookie}, \mathit{filter}) = \mathsf{true} \wedge \kappa_c \sqsubseteq \kappa_1$
Add Cookie to $\mathit{Cookies}_{\mathit{results}_2}$
(6) $\mathit{fresh}(id_{e2})$
$e_2 = (id_{e2}, id_{cb}, \mathsf{none}, \mathit{Cookies}_{\mathit{results}_2}, \kappa_1)$
(7) $\Sigma_2' = \Sigma_2[\Psi_2 \Leftarrow \Psi_2']$
By (A4), (3), (6)
(8) $e_1 \downarrow_\kappa \leq e_2 \downarrow_\kappa$
By STATE, (A5), (3)
(9) $\Sigma_1' \downarrow_\kappa \leq \Sigma_2' \downarrow_\kappa$

sub-subcase ii. $\kappa_1 \not\sqsubseteq \kappa$

By BROWSERSTATE2, Lemma 12
(1) $\psi_1 \downarrow_{\kappa_1} = \cdot$
(2) $\Psi_1' \downarrow_\kappa \leq_c \Psi_1 \downarrow_\kappa$
By STATE, (A5), (2)
(3) $\Sigma_1' \downarrow_\kappa \leq \Sigma_1 \downarrow_\kappa$
By (A4) , EVENT2
(4) $e_1 \downarrow_\kappa = \cdot$

case: COOKIECOREREMOVE(1)

CookieCoreRemove(1) Extension core removes a cookie with name and url. The callback function can access the removed cookie.

$$\frac{\begin{array}{l}\Sigma = (\Psi, \cdots, \mathit{Cookies}, \cdots) \qquad \Psi = \Psi' :: \mathsf{chrome.cookies.remove}(\kappa, id_{cb}, (\mathit{name}, \mathit{url})) \\ \mathit{Cookies} = \mathit{Cookies}' :: \mathit{Cookie} \\ \mathit{Cookie} = (\mathit{name}, \_, \mathit{url}, \kappa_c) \qquad \kappa \sqsubseteq \kappa_c \qquad \Sigma' = \Sigma[\Psi \Leftarrow \Psi'][\mathit{Cookies} \Leftarrow \mathit{Cookies}'] \\ \mathit{fresh}(id_e) \qquad e = (id_e, id_{cb}, \mathsf{none}, \mathit{Cookie}, \kappa_c)\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{chrome.cookies.remove}(\kappa, id_{cb}, (\mathit{name}, \mathit{url}))) = \Sigma'; e}\ \text{COOKIECOREREMOVE(1)}$$

Let $\Sigma_1 = (\Psi_1, \mathit{Tabs}_1, \mathit{ExtCoreRs}_1, \mathit{progInjCSs}_1, \mathit{Exts}_1, \mathit{Cookies}_1, \mathit{MBookmarks}_1, \mathit{histories}_1, \mathit{UI}_1)$
$\Sigma_2 = (\Psi_2, \mathit{Tabs}_2, \mathit{ExtCoreRs}_2, \mathit{progInjCSs}_2, \mathit{Exts}_2, \mathit{Cookies}_2, \mathit{MBookmarks}_2, \mathit{histories}_2, \mathit{UI}_2)$

By assumptions
(A1) $\Psi_1 = \Psi'_1 :: \psi_1$
(A2) $\psi_1 = \mathsf{chrome.cookies.remove}(\kappa_1, id_{cb}, (name, url))$
(A3) $Cookies_1 = Cookies'_1 :: Cookie_1$
(A4) $Cookie_1 = (name, \_, url, \kappa_c)$
(A5) $\kappa_1 \sqsubseteq \kappa_c$
(A6) $\Sigma'_1 = \Sigma_1[\Psi_1 \Leftarrow \Psi'_1][Cookies_1 \Leftarrow Cookies'_1]$
(A7) $fresh(id_{e1})$
(A8) $e_1 = (id_{e1}, id_{cb}, \mathsf{none}, Cookie_1, \kappa_c)$

sub-subcase i. $\kappa_1 \sqsubseteq \kappa$
By BROWSERSTATE1
(1) $\psi_1 \downarrow_\kappa = \psi_1$
By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$, Lemma 11, and (1)
(2) $\Psi_1 \downarrow_\kappa \leq_c \Psi_2 \downarrow_\kappa$
(3) $\exists \psi_2 \in \Psi_2$ s.t. $\psi_1 \downarrow_\kappa \leq \psi_2 \downarrow_\kappa$,
that is $\psi_2 = \psi_1 = \mathsf{chrome.cookies.remove}(\kappa_1, id_{cb}, (name, url))$,
$\Psi_2 = \Psi'_2 :: \psi_2$, $\Psi'_1 \downarrow_\kappa \leq_c \Psi'_2 \downarrow_\kappa$
case 1: $\kappa_c \leq \kappa$
By COOKIE1, $Cookies_1 \downarrow_\kappa \leq Cookies_2 \downarrow_\kappa$
(4) $\exists Cookie_2 \in Cookies_2$ s.t. $Cookie_1 \downarrow_\kappa \leq Cookie_2 \downarrow_\kappa$,
that is $Cookie_1 = Cookie_2$
(5) $Cookies_2 = Cookies'_2 :: Cookie_2$
(6) $Cookies'_1 \downarrow_\kappa \leq Cookies'_2 \downarrow_\kappa$
By (3), (4), apply COOKIECOREREMOVE(1) to Σ_2
(7) $Cookies_2 = Cookies'_2 :: Cookie_2$
(8) $\Sigma'_2 = \Sigma_2[\Psi_2 \Leftarrow \Psi'_2][Cookies_2 \Leftarrow Cookies'_2]$
(9) $fresh(id_{e2})$
(10) $e_2 = (id_{e2}, id_{cb}, \mathsf{none}, Cookie_2, \kappa_c)$
By (A8), (4), (10)
(11) $e_1 \downarrow_\kappa \leq e_2 \downarrow_\kappa$
By STATE, (A6), (3), (6), (8)
(12) $\Sigma'_1 \downarrow_\kappa \leq \Sigma'_2 \downarrow_\kappa$
case 2: $\kappa_c \not\leq \kappa$
By COOKIE2,
(13) $Cookie_1 \downarrow_\kappa = \cdot$
(14) $Cookies'_1 \downarrow_\kappa = Cookies_1 \downarrow_\kappa$
By STATE, (A6), (3), (14)
(15) $\Sigma'_1 \downarrow_\kappa \leq \Sigma_1 \downarrow_\kappa$
By (A8), EVENT2
(16) $e_1 \downarrow_\kappa = \cdot$

sub-subcase ii. $\kappa_1 \not\sqsubseteq \kappa$
By BROWSERSTATE2, Lemma 12
(1) $\psi_1 \downarrow_{\kappa_1} = \cdot$
(2) $\Psi'_1 \downarrow_\kappa \leq_c \Psi_1 \downarrow_\kappa$
By $\kappa_1 \not\sqsubseteq \kappa$, (A5)
(3) $\kappa_c \not\sqsubseteq \kappa$
By (3), COOKIE2
(5) $Cookie_1 \downarrow_\kappa = \cdot$

(6) $\mathit{Cookies}'_1 \downarrow_\kappa \leq \mathit{Cookies}_1 \downarrow_\kappa$

By STATE, (A6), (2), (6)

(7) $\Sigma'_1 \downarrow_\kappa \leq \Sigma_1 \downarrow_\kappa$

By (A8) , (3) , EVENT2

(8) $e_1 \downarrow_\kappa = \cdot$

case: HISTORYSEARCH

$$\frac{\begin{array}{l}\Sigma = (\Psi, \cdots, \mathit{histories}, \cdots) \\ \Psi = \Psi' :: \mathsf{chrome.history.search}(\kappa, \mathit{id}_{cb}, \mathit{query}) \qquad \mathit{histories} = \mathit{histories}_1 :: \mathit{histories}_2 \\ \forall \mathit{history} \in \mathit{histories}_1, \mathit{history} = (\mathit{id}, \mathit{url}, \mathit{name}, \mathit{visitTime}, \mathit{visitType}, \kappa_h) \\ \quad \mathsf{matchQuery}(\mathit{history}, \mathit{query}) = \mathsf{true} \\ \quad \kappa_h \sqsubseteq \kappa \\ \quad \nexists \mathit{history}' \in \mathit{histories}, \mathit{history}' = (\mathit{id}', \mathit{url}, \mathit{name}, \mathit{visitTime}', \mathit{visitType}', \kappa'_h), \mathit{history}' \neq \mathit{history} \\ \qquad \text{s.t. } \mathit{visitTime}' \text{ is later than } \mathit{visitTime} \wedge \kappa'_h \sqsubseteq \kappa \\ \forall \mathit{history} \in \mathit{histories}_2, \mathit{history} = (\mathit{id}, \mathit{url}, \mathit{name}, \mathit{visitTime}, \mathit{visitType}, \kappa_h) \\ \quad \mathsf{matchQuery}(\mathit{history}, \mathit{query}) = \mathsf{true} \\ \quad \kappa_h \sqsubseteq \kappa \\ \quad \exists \mathit{history}' \in \mathit{histories}_1, \mathit{history}' = (\mathit{id}', \mathit{url}, \mathit{name}, \mathit{visitTime}', \mathit{visitType}', \kappa'_h), \\ \qquad \text{s.t. } \mathit{visitTime}' \text{ is later than } \mathit{visitTime} \wedge \kappa'_h \sqsubseteq \kappa \\ \Sigma' = \Sigma[\Psi \Leftarrow \Psi'] \qquad \mathit{fresh}(\mathit{id}_e) \qquad e = (\mathit{id}_e, \mathit{id}_{cb}, \mathsf{none}, \mathit{histories}_1, \kappa)\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{chrome.history.search}(\kappa, \mathit{id}_{cb}, \mathit{query})) = \Sigma'; e}\ \text{HISTORYSEARCH}$$

Let $\Sigma_1 = (\Psi_1, \mathit{Tabs}_1, \mathit{ExtCoreRs}_1, \mathit{progInjCSs}_1, \mathit{Exts}_1, \mathit{Cookies}_1, \mathit{MBookmarks}_1, \mathit{histories}_1, \mathit{UI}_1)$
$\Sigma_2 = (\Psi_2, \mathit{Tabs}_2, \mathit{ExtCoreRs}_2, \mathit{progInjCSs}_2, \mathit{Exts}_2, \mathit{Cookies}_2, \mathit{MBookmarks}_2, \mathit{histories}_2, \mathit{UI}_2)$

By assumptions

(A1) $\Psi_1 = \Psi'_1 :: \psi_1$

(A2) $\psi_1 = \mathsf{chrome.history.search}(\kappa_1, \mathit{id}_{cb}, \mathit{query})$

(A3) $\mathit{histories}_1 = \mathit{histories}_{1a} :: \mathit{histories}_{1b}$

(A4) $\forall \mathit{history} \in \mathit{histories}_{1a}, \mathit{history} = (\mathit{id}, \mathit{url}, \mathit{name}, \mathit{visitTime}, \mathit{visitType}, \kappa_h)$
$\mathsf{matchQuery}(\mathit{history}, \mathit{query}) = \mathsf{true}$
$\kappa_h \sqsubseteq \kappa_1$
$\nexists \mathit{history}' \in \mathit{histories}_1, \mathit{history}' = (\mathit{id}', \mathit{url}, \mathit{name}, \mathit{visitTime}', \mathit{visitType}', \kappa'_h), \mathit{history}' \neq \mathit{history}$
s.t. $\mathit{visitTime}'$ is later than $\mathit{visitTime} \wedge \kappa'_h \sqsubseteq \kappa$

(A5) $\forall \mathit{history} \in \mathit{histories}_{1b}, \mathit{history} = (\mathit{id}, \mathit{url}, \mathit{name}, \mathit{visitTime}, \mathit{visitType}, \kappa_h)$
$\mathsf{matchQuery}(\mathit{history}, \mathit{query}) = \mathsf{true}$
$\kappa_h \sqsubseteq \kappa$
$\exists \mathit{history}' \in \mathit{histories}_{1a}, \mathit{history}' = (\mathit{id}', \mathit{url}, \mathit{name}, \mathit{visitTime}', \mathit{visitType}', \kappa'_h),$
s.t. $\mathit{visitTime}'$ is later than $\mathit{visitTime} \wedge \kappa'_h \sqsubseteq \kappa_1$

(A6) $\Sigma'_1 = \Sigma_1[\Psi_1 \Leftarrow \Psi'_1]$

(A7) $\mathit{fresh}(\mathit{id}_{e1})$, $e_1 = (\mathit{id}_{e1}, \mathit{id}_{cb}, \mathsf{none}, \mathit{histories}_{1a}, \kappa_1)$

sub-subcase i. $\kappa_1 \sqsubseteq \kappa$

By BROWSERSTATE1

(1) $\psi_1 \downarrow_\kappa = \psi_1$

By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$, Lemma 11, and (1)

(2) $\Psi_1 \downarrow_\kappa \leq_c \Psi_2 \downarrow_\kappa$

(3) $\exists \psi_2 \in \Psi_2$ s.t. $\psi_1 \downarrow_\kappa \leq \psi_2 \downarrow_\kappa$,
that is $\psi_2 = \psi_1 = \mathsf{chrome.history.search}(\kappa_1, \mathit{id}_{cb}, \mathit{query})$,

$\Psi_2 = \Psi_2' :: \psi_2$, $\Psi_1' \downarrow_\kappa \leq_c \Psi_2' \downarrow_\kappa$

By (3), HISTORY1

(4) $\forall history \in histories_1$, s.t., $\mathsf{matchQuery}(history, query) = \mathsf{true}$,
$\mathsf{labOf}(history) \sqsubseteq \kappa_1 \sqsubseteq \kappa$
$history \downarrow_\kappa = history$
$history \in histories_1 \downarrow_\kappa$

(5) $\forall history' \in histories_2$, s.t., $\mathsf{matchQuery}(history', query) = \mathsf{true}$,
$\mathsf{labOf}(history') \sqsubseteq \kappa_1 \sqsubseteq \kappa$
$history' \downarrow_\kappa = history'$
$history' \in histories_2 \downarrow_\kappa$

By (4), (5), , $histories_1 \downarrow_\kappa \leq_{hstrs} histories_2 \downarrow_\kappa$, HISTORIES

(6) $histories_1 \downarrow_\kappa = histories_2 \downarrow_\kappa$

By (3), apply HISTORYSEARCH to Σ_2

(7) $histories_2 = histories_{2a} :: histories_{2b}$

(8) $\forall history \in histories_{2a}, history = (id, url, name, visitTime, visitType, \kappa_h)$
$\mathsf{matchQuery}(history, query) = \mathsf{true}$
$\kappa_h \sqsubseteq \kappa_1$
$\nexists history' \in histories_1, history' = (id', url, name, visitTime', visitType', \kappa_h'), history' \neq history$
s.t. $visitTime'$ is later than $visitTime \ \wedge\ \kappa_h' \sqsubseteq \kappa$

(9) $\forall history \in histories_{2b}, history = (id, url, name, visitTime, visitType, \kappa_h)$
$\mathsf{matchQuery}(history, query) = \mathsf{true}$
$\kappa_h \sqsubseteq \kappa$
$\exists history' \in histories_{2a}, history' = (id', url, name, visitTime', visitType', \kappa_h')$,
s.t. $visitTime'$ is later than $visitTime \ \wedge\ \kappa_h' \sqsubseteq \kappa_1$

(10) $\Sigma_2' = \Sigma_2[\Psi_2 \Leftarrow \Psi_2']$

(11) $fresh(id_{e2})$, $e_2 = (id_{e2}, id_{cb}, \mathsf{none}, histories_{2a}, \kappa_1)$

By (A4), (A5), (8), (9), (4), (5), and (6)

(12) $histories_{1a} = histories_{2a}$

By STATE, (A6), (10) and (3)

(13) $\Sigma_1' \downarrow_\kappa \leq \Sigma_2' \downarrow_\kappa$

By (A7), (11), (12)

(14) $e_1 \downarrow_\kappa \leq e_2 \downarrow_\kappa$

sub-subcase ii. $\kappa_1 \not\sqsubseteq \kappa$

By BROWSERSTATE2, Lemma 12

(1) $\psi_1' \downarrow_\kappa = \cdot$

(2) $\Psi_1' \downarrow_\kappa \leq_c \psi_1 \downarrow_\kappa$

By STATE, (A6), and (2)

(3) $\Sigma_1' \downarrow_\kappa \leq \Sigma_1 \downarrow_\kappa$

By (A7), EVENT2

(4) $e_1 \downarrow_\kappa = \cdot$

The proofs for HISTORYGETVISITS are similar to the proofs for this rule.

case: HISTORYADDURL

$$\frac{\begin{array}{l}\Sigma = (\Psi, \cdots, \mathit{histories}, \cdots) \\ \Psi = \Psi' :: \mathsf{chrome.history.addUrl}(\kappa, id_{cb}, (\mathit{name}, \mathit{url}, \mathit{visitTime}, \mathit{visitType})) \\ \mathit{histories}' = \mathit{histories} :: \mathit{history} \\ \mathit{fresh}(id) \qquad \mathit{history} = (id, \mathit{url}, \mathit{name}, \mathit{visitTime}, \mathit{visitType}, \kappa) \\ \Sigma' = \Sigma[\Psi \Leftarrow \Psi'][\mathit{histories} \Leftarrow \mathit{histories}'] \qquad \mathit{fresh}(id_e) \qquad e = (id_e, id_{cb}, \mathsf{none}, \mathit{history}, \kappa)\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{chrome.history.addUrl}(\kappa, id_{cb}, (\mathit{name}, \mathit{url}, \mathit{visitTime}, \mathit{visitType}))) = \Sigma'; e}\ \textsc{historyAddURL}$$

Let $\Sigma_1 = (\Psi_1, \mathit{Tabs}_1, \mathit{ExtCoreRs}_1, \mathit{progInjCSs}_1, \mathit{Exts}_1, \mathit{Cookies}_1, \mathit{MBookmarks}_1, \mathit{histories}_1, \mathit{UI}_1)$
$\Sigma_2 = (\Psi_2, \mathit{Tabs}_2, \mathit{ExtCoreRs}_2, \mathit{progInjCSs}_2, \mathit{Exts}_2, \mathit{Cookies}_2, \mathit{MBookmarks}_2, \mathit{histories}_2, \mathit{UI}_2)$

By assumptions

(A1) $\Psi_1 = \Psi_1' :: \psi_1$
(A2) $\psi_1 = \mathsf{chrome.history.addUrl}(\kappa_1, id_{cb}, (\mathit{name}, \mathit{url}, \mathit{visitTime}, \mathit{visitType}))$
(A3) $\mathit{fresh}(id_1)$, $\mathit{history}_1 = (id_1, \mathit{url}, \mathit{name}, \mathit{visitTime}, \mathit{visitType}, \kappa_1)$
(A4) $\mathit{histories}_1' = \mathit{histories}_1 :: \mathit{history}_1$
(A5) $\Sigma_1' = \Sigma_1[\Psi_1 \Leftarrow \Psi_1'][\mathit{histories}_1 \Leftarrow \mathit{histories}_1']$
(A6) $\mathit{fresh}(id_{e1})$, $e_1 = (id_{e1}, id_{cb}, \mathsf{none}, \mathit{history}_1, \kappa_1)$

sub-subcase i. $\kappa_1 \sqsubseteq \kappa$

By BROWSERSTATE1

(1) $\psi_1 \downarrow_\kappa = \psi_1$

By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$, Lemma 11, and (1)

(2) $\Psi_1 \downarrow_\kappa \leq_c \Psi_2 \downarrow_\kappa$
(3) $\exists \psi_2 \in \Psi_2$ s.t. $\psi_1 \downarrow_\kappa \leq \psi_2 \downarrow_\kappa$,
that is $\psi_2 = \psi_1 = \mathsf{chrome.history.addUrl}(\kappa_1, id_{cb}, (\mathit{name}, \mathit{url}, \mathit{visitTime}, \mathit{visitType}))$,
$\Psi_2 = \Psi_2' :: \psi_2$, $\Psi_1' \downarrow_\kappa \leq_c \Psi_2' \downarrow_\kappa$

By (A3), HISTORY1

(4) $\mathit{history}_1 \downarrow_\kappa = \mathit{history}_1$

Apply HISTORYADDURL to Σ_2

(5) $\mathit{fresh}(id_2)$, $\mathit{history}_2 = (id_2, \mathit{url}, \mathit{name}, \mathit{visitTime}, \mathit{visitType}, \kappa_1)$
(6) $\mathit{histories}_2' = \mathit{histories}_2 :: \mathit{history}_2$
(7) $\Sigma_2' = \Sigma_2[\Psi_2 \Leftarrow \Psi_2'][\mathit{histories}_2 \Leftarrow \mathit{histories}_2']$
(8) $\mathit{fresh}(id_{e2})$, $e_2 = (id_{e2}, id_{cb}, \mathsf{none}, \mathit{history}_2, \kappa_1)$

By (A3), (5), (A4), (6), $\mathit{histories}_1 \downarrow_\kappa \leq_{hstrs} \mathit{histories}_2 \downarrow_\kappa$

(9) $\mathit{history}_1 = \mathit{history}_2$
(10) $\mathit{histories}_1' \leq_{hstrs} \mathit{histories}_2'$

By STATE, (A5), (7), (3), and (10)

(11) $\Sigma_1' \downarrow_\kappa \leq \Sigma_2' \downarrow_\kappa$

By (A6), (8), (9), EVENT1

(12) $e_1 \downarrow_\kappa \leq e_2 \downarrow_\kappa$

sub-subcase ii. $\kappa_1 \not\sqsubseteq \kappa$

By BROWSERSTATE2, Lemma 12

(1) $\psi_1' \downarrow_\kappa = \cdot$
(2) $\Psi_1' \downarrow_\kappa \leq_c \psi_1 \downarrow_\kappa$

By HISTORY2

(3) $\mathit{history}_1 \downarrow_\kappa = \cdot$
(4) $\mathit{histories}_1' \downarrow_\kappa \leq_{hstrs} \mathit{histories}_1 \downarrow_\kappa$

By STATE, (A5), (2), and (4)

(5) $\Sigma_1' \downarrow_\kappa \leq \Sigma_1 \downarrow_\kappa$

By (A6), EVENT2

(6) $e_1 \downarrow_\kappa = \cdot$

case: HISTORYDELETEURL

$$\dfrac{\begin{array}{l} \Sigma = (\Psi, \cdots, \mathit{histories}, \cdots) \\ \Psi = \Psi' :: \mathsf{chrome.history.deleteUrl}(\kappa, \mathit{id}_{cb}, \mathit{url}) \qquad \mathit{histories} = \mathit{histories}_1 :: \mathit{histories}_2 \\ \forall \mathit{history} \in \mathit{histories}_1 \\ \quad \mathit{history} = (\_, \mathit{url}, \_, \_, \_, \kappa_h) \\ \quad \kappa \sqsubseteq \kappa_h \\ \nexists \mathit{history} \in \mathit{histories}_2, \text{ s.t. } \mathit{history} = (\_, \mathit{url}, \_, \_, \_, \kappa_h) \ \wedge \ \kappa \sqsubseteq \kappa_h \\ \Sigma' = \Sigma[\Psi \Leftarrow \Psi'][\mathit{histories} \Leftarrow \mathit{histories}_2] \\ e = (\mathit{id}_e, \mathit{id}_{cb}, \mathsf{none}, \mathsf{void}, \kappa) \end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{chrome.history.deleteUrl}(\kappa, \mathit{id}_{cb}, \mathit{url})) = \Sigma'; e} \ \text{HISTORYDELETEURL}$$

Let $\Sigma_1 = (\Psi_1, \mathit{Tabs}_1, \mathit{ExtCoreRs}_1, \mathit{progInjCSs}_1, \mathit{Exts}_1, \mathit{Cookies}_1, \mathit{MBookmarks}_1, \mathit{histories}_1, \mathit{UI}_1)$
$\Sigma_2 = (\Psi_2, \mathit{Tabs}_2, \mathit{ExtCoreRs}_2, \mathit{progInjCSs}_2, \mathit{Exts}_2, \mathit{Cookies}_2, \mathit{MBookmarks}_2, \mathit{histories}_2, \mathit{UI}_2)$

By assumptions

(A1) $\Psi_1 = \Psi_1' :: \psi_1$

(A2) $\psi_1 = \mathsf{chrome.history.deleteUrl}(\kappa_1, \mathit{id}_{cb}, \mathit{url})$

(A3) $\mathit{histories}_1 = \mathit{histories}_{1a} :: \mathit{histories}_{1b}$

(A4) $\forall \mathit{history} \in \mathit{histories}_{1a}$
$\quad \mathit{history} = (\_, \mathit{url}, \_, \_, \_, \kappa_h)$
$\quad \kappa_1 \sqsubseteq \kappa_h$

(A5) $\nexists \mathit{history} \in \mathit{histories}_{1b}, \text{ s.t. } \mathit{history} = (\_, \mathit{url}, \_, \_, \_, \kappa_h) \ \wedge \ \kappa_1 \sqsubseteq \kappa_h$

(A6) $\Sigma_1' = \Sigma_1[\Psi_1 \Leftarrow \Psi_1'][\mathit{histories}_1 \Leftarrow \mathit{histories}_{1b}]$

(A7) $\mathit{fresh}(\mathit{id}_{e1})$, $e_1 = (\mathit{id}_{e1}, \mathit{id}_{cb}, \mathsf{none}, \mathsf{void}, \kappa_1)$

sub-subcase i. $\kappa_1 \sqsubseteq \kappa$

By BROWSERSTATE1

(1) $\psi_1 \downarrow_\kappa = \psi_1$

By $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$, Lemma 11, and (1)

(2) $\Psi_1 \downarrow_\kappa \leq_c \Psi_2 \downarrow_\kappa$

(3) $\exists \psi_2 \in \Psi_2$ s.t. $\psi_1 \downarrow_\kappa \leq \psi_2 \downarrow_\kappa$,
that is $\psi_2 = \psi_1 = \mathsf{chrome.history.deleteUrl}(\kappa_1, \mathit{id}_{cb}, \mathit{url})$,
$\Psi_2 = \Psi_2' :: \psi_2$, $\Psi_1' \downarrow_\kappa \leq_c \Psi_2' \downarrow_\kappa$

Apply HISTORYDELETEURL to Σ_2

(4) $\mathit{histories}_2 = \mathit{histories}_{2a} :: \mathit{histories}_{2b}$

(5) $\forall \mathit{history} \in \mathit{histories}_{2a}$
$\quad \mathit{history} = (\_, \mathit{url}, \_, \_, \_, \kappa_h)$
$\quad \kappa_1 \sqsubseteq \kappa_h$

(6) $\nexists \mathit{history} \in \mathit{histories}_{2b}, \text{ s.t. } \mathit{history} = (\_, \mathit{url}, \_, \_, \_, \kappa_h) \ \wedge \ \kappa_1 \sqsubseteq \kappa_h$

(7) $\Sigma_2' = \Sigma_2[\Psi_2 \Leftarrow \Psi_2'][\mathit{histories}_2 \Leftarrow \mathit{histories}_{2b}]$

(8) $\mathit{fresh}(\mathit{id}_{e2})$, $e_2 = (\mathit{id}_{e2}, \mathit{id}_{cb}, \mathsf{none}, \mathsf{void}, \kappa_1)$

By (A4), (5), HISTORY1, HISTORY2, $\mathit{histories}_1 \downarrow_\kappa \leq_{hstrs} \mathit{histories}_2 \downarrow_\kappa$

(9) $\mathit{histories}_1 \downarrow_\kappa = \mathit{histories}_2 \downarrow_\kappa$

(10) $\forall \mathit{history} \in \mathit{histories}_{1a}$, s.t., $\mathsf{labOf}(\mathit{history}) \sqsubseteq \kappa$
$\quad \exists \mathit{history}' \in \mathit{histories}_{2a}$, s.t., $\mathit{history} \downarrow_\kappa = \mathit{history}' \downarrow_\kappa$

(11) $\forall \mathit{history} \in \mathit{histories}_{2a}$, s.t., $\mathsf{labOf}(\mathit{history}) \sqsubseteq \kappa$

$\exists \mathit{history}' \in \mathit{histories}_{1a}$, s.t., $\mathit{history} \downarrow_\kappa = \mathit{history}' \downarrow_\kappa$

(12) $\mathit{histories}_{1a} \downarrow_\kappa = \mathit{histories}_{2a} \downarrow_\kappa$

(13) $\mathit{histories}_{1b} \downarrow_\kappa = \mathit{histories}_{2b} \downarrow_\kappa$

(14) $\mathit{histories}_{1b} \downarrow_\kappa \leq_{hstrs} \mathit{histories}_{2b} \downarrow_\kappa$

By STATE, (A6), (7), (3), and (14)

(11) $\Sigma_1' \downarrow_\kappa \leq \Sigma_2' \downarrow_\kappa$

By (A7), (8), EVENT1

(12) $e_1 \downarrow_\kappa \leq e_2 \downarrow_\kappa$

sub-subcase ii. $\kappa_1 \not\sqsubseteq \kappa$

By BROWSERSTATE2, Lemma 12

(1) $\psi_1' \downarrow_\kappa = \cdot$

(2) $\Psi_1' \downarrow_\kappa \leq_c \psi_1 \downarrow_\kappa$

By HISTORY2 , HISTORIES

(3) $\mathit{history}_{1a} \downarrow_\kappa = \cdot$

(4) $\mathit{histories}_{1b} \downarrow_\kappa \leq_{hstrs} \mathit{histories}_1 \downarrow_\kappa$

By STATE, (A6), (2), and (4)

(5) $\Sigma_1' \downarrow_\kappa \leq \Sigma_1 \downarrow_\kappa$

By (A7), EVENT2

(6) $e_1 \downarrow_\kappa = \cdot$

case: NEWTAB

$$\frac{\begin{array}{l} \mathit{fresh}(id_t, id_d, id_{e1}, id_{e2}) \qquad \Sigma = (\Psi, \mathit{Tabs}, \cdots) \\ \Psi = \Psi' :: \mathsf{chrome.tabs.create}(\kappa, id_{cb}, (id_w, \mathit{positionIndex}, \mathit{url}, \mathit{activeFlag}, \mathit{pinnedBoolean}, id_{parentTab})) \\ t = (id_t, \cdot, \mathit{url}, \cdot, \cdot, \kappa^{FC}) \qquad \psi' = \mathsf{readyToFetchDoc}(id_t, id_d, \cdot, id_d, \mathit{url}) \\ \Sigma' = \Sigma[\Psi \Leftarrow \Psi' :: \psi][\mathit{Tabs} \Leftarrow \mathit{Tabs} :: t] \qquad \mathcal{E} = (id_{e1}, \mathsf{tabs.onCreated}, \mathsf{none}, \mathit{info}_1, \kappa) \\ \quad :: (id_{e2}, id_{cb}, \mathsf{none}, \mathsf{getInfo}(t), \kappa) \end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{chrome.tabs.create}(\kappa, id_{cb}, (id_w, \mathit{positionIndex}, \mathit{url}, \mathit{activeFlag}, \mathit{pinnedBoolean}, id_{parentTab}))) = \Sigma', \mathcal{E}} \text{ NEWTAB}$$

Let $\Sigma_1 = (\Psi_1, \mathit{Tabs}_1, \mathit{ExtCoreRs}_1, \mathit{progInjCSs}_1, \mathit{Exts}_1, \mathit{Cookies}_1, \mathit{MBookmarks}_1, \mathit{histories}_1, \mathit{UI}_1)$

$\Sigma_2 = (\Psi_2, \mathit{Tabs}_2, \mathit{ExtCoreRs}_2, \mathit{progInjCSs}_2, \mathit{Exts}_2, \mathit{Cookies}_2, \mathit{MBookmarks}_2, \mathit{histories}_2, \mathit{UI}_2)$

By assumptions

(A1) $\mathit{fresh}(id_t, id_d, id_{e1}, id_{e2})$

(A2) $\Psi_1 = \Psi_1'' :: \psi_1$

$\psi_1 = \mathsf{chrome.tabs.create}(\kappa_1, id_{cb}, (id_w, \mathit{positionIndex}, \mathit{url}, \mathit{activeFlag}, \mathit{pinnedBoolean}, id_{parentTab}))$

(A3) $t_1 = (id_t, \cdot, \mathit{url}, \cdot, \cdot, \kappa_1^{FC})$

(A4) $\psi_1' = \mathsf{readyToFetchDoc}(\kappa, id_t, id_d, \cdot, id_d, \mathit{url})$

(A5) $\Sigma_1' = \Sigma_1[\Psi_1 \Leftarrow \Psi_1'' :: \psi_1'][\mathit{Tabs}_1 \Leftarrow \mathit{Tabs}_1 :: t_1]$

(A6) $\mathcal{E}_1 = (id_{e1}, \mathsf{tabs.onCreated}, \mathsf{none}, \mathit{info}_1, \kappa_1) :: (id_{e2}, id_{cb}, \mathsf{none}, \mathsf{getInfo}(t), \kappa_1)$

sub-subcase i. $\kappa_1 \sqsubseteq \kappa$

By BROWSERSTATE1

(1) $\psi_1 \downarrow_\kappa = \psi_1, \psi_1 \in \Psi_1 \downarrow_\kappa$

By $\Psi_1 \downarrow_\kappa \leq_c \Psi_2 \downarrow_\kappa$, (1)

(2) $\exists \psi_2 \in \Psi_2 \downarrow_\kappa$, s.t. $\psi_1 \downarrow_\kappa \leq \psi_2 \downarrow_\kappa$, namely $\psi_2 = \psi_1$

Apply NEWTAB to Σ_2

(3) $\psi_2 = \psi_1 = \mathsf{chrome.tabs.create}(\kappa_1, id_{cb}, (id_w, \mathit{positionIndex}, \mathit{url}, \mathit{activeFlag}, \mathit{pinnedBoolean}, id_{parentTab}))$

(4) $\mathit{fresh}(id_{t2}, id_{d2}, id_{e1}', id_{e2}') = \mathit{fresh}(id_t, id_d, id_{e1}, id_{e2})$

(5) $t_2 = (id_{t2}, \cdot, \mathit{url}, \cdot, \cdot, \kappa_1^{FC}) = t_1$

(6) $\psi_2' = \mathsf{readyToFetchDoc}(id_{t2}, id_{d2}, \cdot, id_{d2}, url) = \psi_1'$

(7) $\Sigma_2' = \Sigma_2[\Psi_s \Leftarrow \Psi_2'' :: \psi_2'][Tabs_2 \Leftarrow Tabs_2 :: t_1]$

(8) $\mathcal{E}_2 = (id_{e}{}_1', \mathsf{tabs.onCreated}, \mathsf{none}, info_1, \kappa_1) :: (id_{e}{}_2', id_{cb}, \mathsf{none}, \mathsf{getInfo}(t_2), \kappa_1) = \mathcal{E}_1$

By Lemma 11

(9) $\Psi_1'' \downarrow_\kappa \leq_c \Psi_2'' \downarrow_\kappa$

By (6), (9), 10

(10) $(\Psi_1'' :: \psi_1') \downarrow_\kappa \leq_c (\Psi_2'' :: \psi_2') \downarrow_\kappa$

By (5), $Tabs_1 \downarrow_\kappa \leq Tabs_2 \downarrow_\kappa$

(11) $Tabs_1 :: t_1 \downarrow_\kappa \leq Tabs_2 :: t_2 \downarrow_\kappa$

By (10),(11), (A5), (7), STATE

(12) $\Sigma_1' \downarrow_\kappa \leq \Sigma_2' \downarrow_\kappa$

By (8)

(13) $\mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$

sub-subcase ii. $\kappa_1 \not\sqsubseteq \kappa$

By (A3), TAB3

(1) $t_1 \downarrow_\kappa = \cdot$

(2) $Tabs_1 :: t_1 \downarrow_\kappa \leq Tabs_1 \downarrow_\kappa$

By (A2), (A4), Lemma 11, Lemma 10

(3) $\Psi_1'' :: \psi_1' \downarrow_\kappa \leq_c \Psi_1 \downarrow_\kappa$

By (2), (3), (A5), STATE

(4) $\Sigma_1' \downarrow_\kappa \leq \Sigma_1 \downarrow_\kappa$

By (A6), EVENT2

(5) $\mathcal{E}_1 \downarrow_\kappa = \cdot$

case: EABLEDISABLEEXT1(A)

$$\frac{\begin{array}{l}\Sigma = (\Psi, Tabs, Exts :: Ext, ExtCoreRs, \cdots)\\ \Psi = \Psi' :: \mathsf{chrome.management.setEnabled}(\kappa, id_{cb}, (string, id, id_{ext}, activeFlag))\\ Ext = (id_{ext}, ExtCore, \cdots, activeFlag, \ell_1) \qquad \kappa \not\sqsubseteq {\ell_1}^*\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{chrome.management.setEnabled}(\kappa, id_{cb}, (string, id, id_{ext}, activeFlag))) = \Sigma[\Psi \Leftarrow \Psi'], \cdot}\ \text{ENABLEDISABLEEXT1(A)}$$

Let $\Sigma_1 = (\Psi_1, Tabs_1, ExtCoreRs_1, progInjCSs_1, Exts_1, Cookies_1, MBookmarks_1, histories_1, UI_1)$
$\Sigma_2 = (\Psi_2, Tabs_2, ExtCoreRs_2, progInjCSs_2, Exts_2, Cookies_2, MBookmarks_2, histories_2, UI_2)$

By assumptions

(A1) $Ext_1 = (id_{ext}, ExtCore_1, ExtCSs_1, Storage_1, \chi, activeFlag, \ell)$

(A2) $Ext_1 \in Exts_1$

(A3) $\Psi_1 = \Psi_1'' :: \psi_1$

(A4) $\kappa_1 \not\sqsubseteq {\ell_1}^*$

(A5) $\psi_1 = \mathsf{chrome.management.setEnabled}(\kappa_1, id_{cb}, (string, id, id_{ext}, activeFlag))$

(A6) $\Sigma_1' = \Sigma_1[\Psi_1 \Leftarrow \Psi_1'']$

(A7) $\mathcal{E}_1' = \cdot$

sub-subcase i. $\kappa_1 \sqsubseteq \kappa, {\ell_1}^- \sqsubseteq \kappa$

By BROWSERSTATE1

(1) $\psi_1 \downarrow_\kappa = \psi_1, \psi_1 \in \Psi_1 \downarrow_\kappa$

By $\Psi_1 \downarrow_\kappa \leq_c \Psi_2 \downarrow_\kappa$, (1)

(2) $\exists \psi_2 \in \Psi_2 \downarrow_\kappa$, s.t. $\psi_1 \downarrow_\kappa \leq \psi_2 \downarrow_\kappa$, namely $\psi_2 = \psi_1$

(3) $\Psi_2 = \Psi_2'' :: \psi_2$

By $Exts_1 \downarrow_\kappa \leq Exts_2 \downarrow_\kappa$

(4) $\exists Ext_2 \in Exts_2$

s.t. $Ext_2 = (id_{ext}, ExtCore_2, \cdots, activeFlag, \ell_1)$, $ExtCore_1 \leq ExtCore_2$

By (2), (4), apply ENABLEDISABLEEXT1(A) to Σ_2

(5) $\Sigma_2' = \Sigma_2[\Psi_2 \Leftarrow \Psi_2'']$, $\mathcal{E}_2 = \cdot$

By (A3), (2), (3), $\Psi_1 \downarrow_\kappa \leq_c \Psi_2 \downarrow_\kappa$, Lemma 11

(6) $\Psi_1'' \downarrow_\kappa \leq_c \Psi_2'' \downarrow_\kappa$

By STATE, (A6), (5)

(7) $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

By (A7), (5)

(8) $\mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$

sub-subcase ii. $\kappa_1 \sqsubseteq \kappa$, ${\ell_1}^- \not\sqsubseteq \kappa$

By Lemma 11

(1) $\Psi_1'' \downarrow_\kappa \leq_c \Psi_1 \downarrow_\kappa$

By (A6),(1), and STATE

(2) $\Sigma_1' \downarrow_\kappa \leq \Sigma_1 \downarrow_\kappa$

By (2) and $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

(3) $\Sigma_1' \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

By (A7)

(4) $\mathcal{E}_1' \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$

sub-subcase iii. $\kappa_1 \not\sqsubseteq \kappa$,

By BROWSERSTATE2

(1) $\psi_1 \downarrow_\kappa = \cdot$

(2) $\Psi_1 \downarrow_\kappa = \Psi_1'' \downarrow_\kappa$

By (A6)

(3) $\Sigma_1' \downarrow_\kappa = \Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

By (A7)

(4) $\mathcal{E}_1' \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$

case: EABLEDISABLEEXT1(B)

$$\frac{\begin{array}{l}\Sigma = (\Psi, Tabs, Exts :: Ext, ExtCoreRs, \cdots) \\ \Psi = \Psi' :: \mathsf{chrome.management.setEnabled}(\kappa, id_{cb}, (string, id, id_{ext}, activeFlag)) \\ Ext = (id_{ext}, ExtCore, \cdots, activeFlag, \ell_1) \qquad \kappa \sqsubseteq {\ell_1}^* \\ \ell_2 = \kappa \rhd_{tnt} \ell_1 \qquad fresh(id_e) \qquad e_1 = (id_e, id_{cb}, \mathsf{none}, \mathsf{getInfo}(Ext), {\ell_2}^-)\end{array}}{\mathsf{nextStateC}(\Sigma, \mathsf{chrome.management.setEnabled}(\kappa, id_{cb}, (string, id, id_{ext}, activeFlag))) = \Sigma[\Psi \Leftarrow \Psi'], e_1} \text{ ENABLEDISABLEEXT1(B}$$

Let $\Sigma_1 = (\Psi_1, Tabs_1, ExtCoreRs_1, progInjCSs_1, Exts_1, Cookies_1, MBookmarks_1, histories_1, UI_1)$

$\Sigma_2 = (\Psi_2, Tabs_2, ExtCoreRs_2, progInjCSs_2, Exts_2, Cookies_2, MBookmarks_2, histories_2, UI_2)$

By assumptions

(A1) $Ext_1 = (id_{ext}, ExtCore_1, \cdots, activeFlag, \ell_1)$

(A2) $Ext_1 \in Exts_1$

(A3) $\Psi_1 = \Psi_1'' :: \psi_1$, $\psi_1 = \mathsf{chrome.management.setEnabled}(\kappa_1, id_{cb}, (string, id, id_{ext}, activeFlag))$

(A4) $\kappa_1 \sqsubseteq {\ell_1}^*$

(A5) $\ell_1' = \kappa \rhd_{tnt} \ell_1$

(A6) $fresh(id_e)$

(A7) $e_1 = (id_e, id_{cb}, \mathsf{none}, \mathsf{getInfo}(Ext_1), {\ell_1'}^-)$

(A8) $\Sigma_1' = \Sigma_1[\Psi_1 \Leftarrow \Psi_1'']$

(A9) $\mathcal{E}_1 = e_1$

sub-subcase i. $\kappa_1 \sqsubseteq \kappa$, ${\ell_1}^- \sqsubseteq \kappa$

By BROWSERSTATE1

(1) $\psi_1 \downarrow_\kappa = \psi_1, \psi_1 \in \Psi_1 \downarrow_\kappa$

By $\Psi_1 \downarrow_\kappa \leq_c \Psi_2 \downarrow_\kappa$, (1)

(2) $\exists \psi_2 \in \Psi_2 \downarrow_\kappa$, s.t. $\psi_1 \downarrow_\kappa \leq \psi_2 \downarrow_\kappa$, namely $\psi_2 = \psi_1$
$\psi_2 = \mathsf{chrome.management.setEnabled}(\kappa_1, id_{cb}, (string, id, id_{ext}, activeFlag))$

(3) $\Psi_2 = \Psi_2'' :: \psi_2$

By $Exts_1 \downarrow_\kappa \leq Exts_2 \downarrow_\kappa$

(4) $\exists Ext_2 \in Exts_2$
s.t. $Ext_2 = (id_{ext}, ExtCore_2, \cdots, activeFlag, \ell_1)$, $ExtCore_1 \leq ExtCore_2$

By (A4), (A5), (2), (4), apply ENABLEDISABLEEXT1(B) to Σ_2

(5) $fresh({id_e}')$

(6) $e_2 = ({id_e}', id_{cb}, \mathsf{none}, \mathsf{getInfo}(Ext_2), {\ell_1'}^-) = e_1$

(7) $\Sigma_2' = \Sigma_2[\Psi_2 \Leftarrow \Psi_2'']$

(8) $\mathcal{E}_2 = e_2$

By (A3), (2), (3), $\Psi_1 \downarrow_\kappa \leq_c \Psi_2 \downarrow_\kappa$, Lemma 11

(9) $\Psi_1'' \downarrow_\kappa \leq_c \Psi_2'' \downarrow_\kappa$

By STATE, (A8), (7), (9)

(10) $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

By (A9), (6), (8)

(11) $\mathcal{E}_1 \downarrow_\kappa \leq \mathcal{E}_2 \downarrow_\kappa$

sub-subcase ii. $\kappa_1 \sqsubseteq \kappa$, ${\ell_1}^- \not\sqsubseteq \kappa$

By Lemma 11

(1) $\Psi_1'' \downarrow_\kappa \leq_c \Psi_1 \downarrow_\kappa$

By (A8),(1), and STATE

(2) $\Sigma_1' \downarrow_\kappa \leq \Sigma_1 \downarrow_\kappa$

By (2) and $\Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

(3) $\Sigma_1' \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

By EVENT2, (A9)

(4) $\mathcal{E}_1 \downarrow_\kappa = \cdot$

sub-subcase iii. $\kappa_1 \not\sqsubseteq \kappa$

By BROWSERSTATE2

(1) $\psi_1 \downarrow_\kappa = \cdot$

(2) $\Psi_1 \downarrow_\kappa = \Psi_1'' \downarrow_\kappa$

By (A4)

(3) $\Sigma_1' \downarrow_\kappa = \Sigma_1 \downarrow_\kappa \leq \Sigma_2 \downarrow_\kappa$

By (A5), $\kappa_1 \not\sqsubseteq \kappa$, EVENT2

(4) ${\ell_2}^- \not\sqsubseteq \kappa$

(5) $\mathcal{E}_1 \downarrow_\kappa = e_1 \downarrow_\kappa = \cdot$

References

[1] D. Devriese and F. Piessens. Noninterference through secure multi-execution. In *Proc. IEEE S&P*, 2010.

[2] A. C. Myers. Practical mostly-static information flow control. In *Proc. POPL*, 1999.